ISSN 0252–9742

Bulletin

of the

European Association for

Theoretical Computer Science

EATCS

Number 142 February 2024

Council of the European Association for Theoretical Computer Science

President:	Artur Czumaj	United Kingdom
Vice Presidents:	Anca Muscholl	France
	Giuseppe F. Italiano	Italy
Treasurer:	Jean-Francois Raskin	Belgium
Bulletin Editor:	Stefan Schmid	Germany

Antoine Amarilli	France	Giuseppe F. Italiano	Italy
Ivona Bezakova	USA	Emanuela Merelli	Italy
Tiziana Calamoneri	Italy	Anca Muscholl	France
Thomas Colcombet	France	Charles Paperman	French
Artur Czumaj	UK	Tal Rabin	USA
Anne Driemel	Germany	Eva Rotenberg	Denmark
Javier Esparza	Germany	Jiri Sgall	Czech Republic
Inge Li Goertz	Denmark	Jukka Suomela	Finland
Fabrizio Grandoni	Switzerland	Szymon Torunczyk	Poland
Thore Husfeldt	Sweden, Denmark	Bianca Truthe	Germany

Past Presidents:

Maurice Nivat	(1972–1977)	Mike Paterson	(1977–1979)
Arto Salomaa	(1979–1985)	Grzegorz Rozenberg	(1985–1994)
Wilfred Brauer	(1994–1997)	Josep Díaz	(1997–2002)
Mogens Nielsen	(2002–2006)	Giorgio Ausiello	(2006–2009)
Burkhard Monien	(2009–2012)	Luca Aceto	(2012–2016)
Paul Spirakis	(2016–2020)		

Secretary Office:	Dmitry Chistikov	UK
	Efi Chita	Greece

Bulletin Editor: Stefan Schmid, Berlin, Germany
Cartoons: DADARA, Amsterdam, The Netherlands

The bulletin is entirely typeset by pdfTEX and ConTEXt in TXfonts.

All contributions are to be sent electronically to

`bulletin@eatcs.org`

and must be prepared in LaTeX 2$_\varepsilon$ using the class `beatcs.cls` (a version of the standard LaTeX 2$_\varepsilon$ `article` class). All sources, including figures, and a reference PDF version must be bundled in a ZIP file.

Pictures are accepted in EPS, JPG, PNG, TIFF, MOV or, preferably, in PDF. Photographic reports from conferences must be arranged in ZIP files layed out according to the format described at the Bulletin's web site. Please, consult `http://www.eatcs.org/bulletin/howToSubmit.html`.

We regret we are unfortunately not able to accept submissions in other formats, or indeed submission not *strictly* adhering to the page and font layout set out in `beatcs.cls`. We shall also not be able to include contributions not typeset at camera-ready quality.

The details can be found at `http://www.eatcs.org/bulletin`, including class files, their documentation, and guidelines to deal with things such as pictures and overfull boxes. When in doubt, email `bulletin@eatcs.org`.

Deadlines for submissions of reports are January, May and September 15th, respectively for the February, June and October issues. Editorial decisions about submitted technical contributions will normally be made in 6/8 weeks. Accepted papers will appear in print as soon as possible thereafter.

The Editor welcomes proposals for surveys, tutorials, and thematic issues of the Bulletin dedicated to currently hot topics, as well as suggestions for new regular sections.

The EATCS home page is `http://www.eatcs.org`

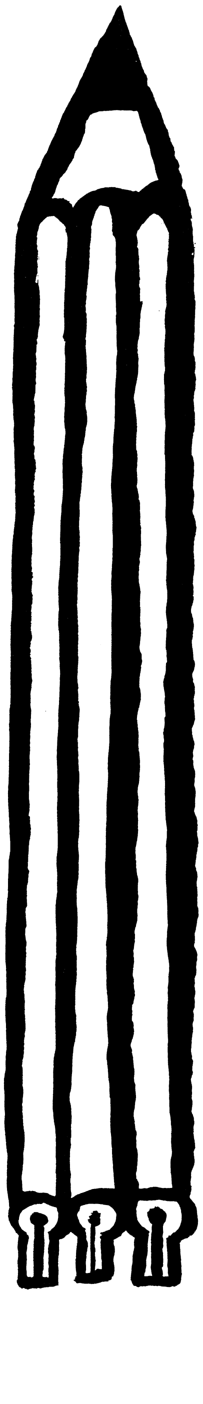
Table of Contents

EATCS Matters

Dear EATCS members,

First of all, let me wish you a very happy 2024. I hope that this will be a healthy and fantastic year for all of us, full of great research advances, exciting conferences and workshops. I look forward to working together with all of you in order to continue promoting the development of theoretical computer science.

I am especially looking forward to attending the 51st EATCS International Colloquium on Automata, Languages, and Programming (ICALP 2024), the EATCS flagship conference that will be held in Tallinn, Estonia, July 8-12, 2024 (https://compose.ioc.ee/icalp2024/). The PC chairs are Karl Bringmann, Ola Svensson (track A), and Martin Grohe (track B), and the conference chair is Pawel Sobocinski. ICALP 2024 will feature three fantastic invited speakers: Anuj Dawar (University of Cambridge), Danupon Nanongkai (MPI Saarbrücken), and Merav Parter (Weizmann Institute), and further two joint (with LICS and FSCD) invited speakers Edith Elkind (University of Oxford) and Stephanie Weirich (University of Pennsylvania). I hope that many of you have submitted your very best work to ICALP 2024 and I expect to see a great scientific program, to be selected by the PC in mid April. As usual, ICALP will be preceded by a series of workshops, which will take place on July 7. ICALP 2024 will be collocated with LICS (39th Annual ACM/IEEE Symposium on Logic in Computer Science) and FSCD (9th International Conference on Formal Structures for Computation and Deduction).



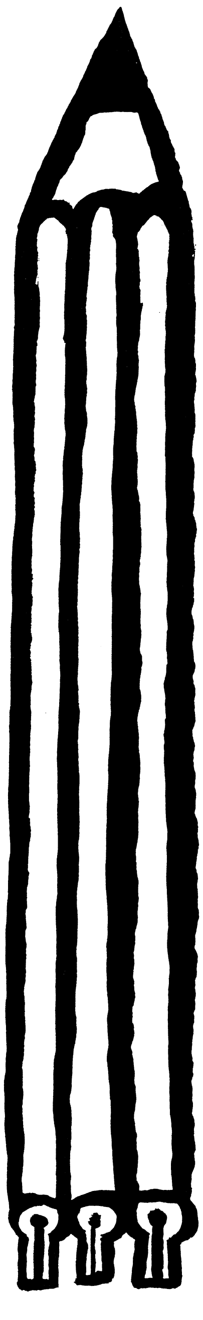
Please pencil these dates in your diary and I hope to see many of you attending the next ICALP in Tallinn.

Also, please allow me to remind you about three EATCS affiliated conferences that will take place later in summer and fall this year: the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024, http://www.mfcs.sk/) in Bratislava, Slovakia, August 26-30, the 32nd Annual European Symposium on Algorithms (ESA 2024, https://algo-conference.org/2024/esa/) at Royal Holloway, University of London in Egham, United Kingdom, September 2-4, and the 37th International Symposium on Distributed Computing (DISC 2024, http://www.disc-conference.org/wp/disc2024/) in Madrid, Spain, October 28-November 1.

But there will be some more exciting theory conferences taking place in the summer, where I hope to see strong in-person attendance stimulating fantastic research advances in theory.

As usual, let me close this letter by reminding you that you are always most welcome to send me your comments, criticisms and suggestions for improving the impact of the EATCS on the Theoretical Computer Science community at president@eatcs.org. We will consider all your suggestions and criticisms carefully.

I look forward to seeing many of you around, at ICALP in Tallinn, or during other conferences or workshops that I hope to attend this spring and summer and fall, or maybe only online, and to discussing ways of improving the impact of the EATCS within the theoretical computer science



community.

Artur Czumaj
University of Warwick, UK
President of EATCS
`president@eatcs.org`

February 2024

Dear EATCS member!

The first edition of the Bulletin this year includes two interviews: Laura Kovacs and Moshe Vardi tell us which papers motivated them to become researchers in their respective fields, give advice on how to deal with failures, and share with us many interesting anecdotes and perspectives about their daily work and about the directions they believe our community may evolve.

The Distributed Computing Column features Siddhartha Jayanti, the winner of the 2023 Principles of Distributed Computing Doctoral Dissertation Award. His thesis presents several important concurrent algorithms, and raises exciting opportunities for simple machine-verified proofs for concurrent data structures.

In the Education Column, Marko Schmellenkamp, Fabian Vehlken, and Thomas Zeume discuss the challenges involved in teaching introductory courses on formal foundations of computer science, including the large class sizes with students with different backgrounds. They report on their positive experiences with supplementing traditional teaching with web-based, interactive exercises, and in particular, the teaching support system Iltis.

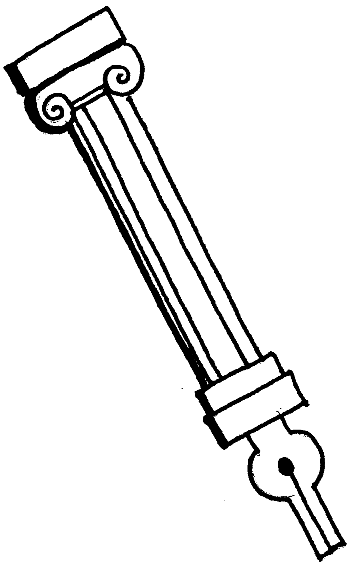
The Bulletin further includes conference reports from the International Workshop on Non-Classical Models of Automata and Applications (NCMA) and from the 27th International Conference on Implementation and Application of Automata (CIAA).

I wish you a happy Spring time and I hope you enjoy the new Bulletin!

Stefan Schmid, Berlin
February 2024

Institutional Sponsors

EATCS Columns



The Interview Column

by

Chen Avin and Stefan Schmid

Ben Gurion University, Israel and TU Berlin, Germany
{chenavin,schmiste}@gmail.com

Know the Person behind the Papers

Today: Laura Kovács

Bio: *Laura Kovács is a full professor of computer science at the TU Wien, leading the automated program reasoning (APRe) group of the Formal Methods in Systems Engineering division. Her research focuses on the design and development of new theories, technologies, and tools for program analysis, with a particular focus on automated assertion generation, symbolic summation, computer algebra, and automated theorem proving. She is the co-developer of the Vampire theorem prover and a Wallenberg Academy Fellow of Sweden. Her research has also been awarded with a ERC Starting Grant 2014, an ERC Proof of Concept Grant 2018, an ERC Consolidator Grant 2020, and an Amazon Research Award 2020. Recently, she received financial support from Let's Empower Austria - LEA Frauenfonds to disseminate computer science to elementary schools.*

EATCS: We ask all interviewees to share a photo with us. Can you please tell us a little bit more about the photo you shared?

LK: The picture is taken in one of my very recent and dearest activities at the TU Wien: introducing elementary school children to the art and fun of computer science. I do various puzzle solving activities with kids and develop algorithms together with them. It is energizing to see how creative and fast kids can be - either in sorting, path finding or instructing small robots. We all have a lot of fun and I hope most of the kids will stick to STEM education - at least they say so with excitement when they leave our workshops!

EATCS: Can you please tell us something about you that probably most of the readers of your papers don't know?

LK: I did my bachelor studies in image processing. I was working on optical music recognition and reconstructed music sounds from printed music sheets. It was a fun project where I did a lot of coding and used many linear algebra algorithms. I was fascinated by the topic for almost two years, before I entered the field of symbolic computation.

EATCS: Is there a paper which influenced you particularly, and which you recommend other community members to read?

LK: I cannot really name one single research paper that influenced me the most. However, the paper that pushed me to start work in formal verification is the master thesis "Program Verification with the Mathematical Software System Theorema" by Martin Kirchner, RISC-Linz, Austria, 1999 (Technical Report 99-16). I read this thesis while being a master student at RISC-Linz and I liked the combination of logic, math and software engineering. Thanks to this thesis, I read the seminal works of Edgar Dijkstra, Robert Floyd, and Tony Hoare - as well as many other related papers and works.

EATCS: Is there a paper of your own you like to recommend the readers to study? What is the story behind this paper?

LK: I particularly like my TACAS 2008 paper "Reasoning Algebraically About P-Solvable Loops". This paper summarizes my PhD thesis and it was written in a quite stressful situation. I actually wanted to prove a stronger result than the one presented in the paper: essentially, my goal was to prove that the strongest inductive invariants of so-called polynomial-solvable loops are computable. However, I could only prove this under additional restrictions on the loop semantics and was quite disappointed that time that I could not get stronger theoretical results. It was a tricky situation: I could not complete the proof of the general case, but I also did not succeed in finding a counterexample. As such, in my TACAS 2008 paper I posed the general case as an open challenge for the future. I am glad I have done so because since then quite many undecidable and/or hardness results have been established upon algebraic invariant synthesis. Notably, in our POPL 2024 on "Strong Invariants Are Hard", we have just proved that generalizing my PhD thesis results is not trivial, hitting the computational limits of the famous Skolem problem from number theory. Even more than 15 years after my PhD, the open challenge of my TACAS 2008 paper brings in new research directions to further explore.

EATCS: When (or where) is your most productive working time (or place)?

LK: Even though I dislike waking up early, early morning hours are the best for me. After dropping my kids at school, I usually have 8-10am as the two hours in which I am free to do what I want, on a daily basis. I actually enjoy working late, but this I can only do for limited periods, triggered by deadlines. In addition, I have one day a week where I have no meetings, lectures or other events. I enjoy this day usually in my office.

EATCS: What do you do when you get stuck with a research problem? How do you deal with failures?

LK: I try to read up more on related work and find a colleague to discuss and brainstorm. Failure is relative and is part of research. I would therefore not really say that I/we failed on a research topic, but consider that the chosen methodology did not work as expected. In such cases, I try to understand what and why was different; if I manage to do so, then I failed "well" and have a very good learning outcome.

EATCS: Is there a nice anecdote from your career you would like to share with our readers?

LK: I come from the Hungarian minority of Transylvania, Romania. I manage to confuse people all the time when I stress that I have double citizenship (Romanian and Hungarian) but only one nationality (Hungarian). Quite many administrative forms do not distinguish between citizenship and nationality, so I end up writing a short explanatory text on a form where most likely only one word is needed.

Funnily, when I mention Transylvania, many people associate my background with vampires and Dracula. When I then tell them that I work on a research project called Vampire, they just believe this has been my destiny. It was however quite a coincidence that I started working on the Vampire theorem prover in 2009.

EATCS: Do you have any advice for young researchers? In what should they invest time, what should they avoid?

LK: There are many advices one gets and one should filter out among these advices. Actually, it is impossible to follow all advices. The one that I follow (and tell my students) is rightfully generic: Do what you are the best at! One should not just follow hypes, rather be persistent on own interests. Positioning these interests within trendy research topics is beneficial, but should not be your driving wheel.

EATCS: What are the most important features you look for when searching for graduate students?

LK: Curiosity and flexibility. One should be passionate about curiosity-driven research and be open to new challenges and collaborations. Most research in computer science is collaborative and one should be respectful to colleagues.

It can be hard for students to assess whether they are "ready" for research. I therefore actively approach some of my best students from my lectures at the TU Wien. I talk to them, invite them to group meetings, and try to initiate joint research topics of mutual interest.

EATCS: Do you see a main challenge or opportunity for theoretical computer scientists for the near future?

LK: I see the opportunity of a continuously evolving, hard research field. There are always new problems that need new solutions; once a problem is solved, we are happy to take up new, harder problems. It is like an endless loop: if one is passionate about research, one never gets bored but remains eternally happy in solving problems. Theoretical computer science is a safe place to be in, it will always exist and will always be challenged by new computer science applications and practices.

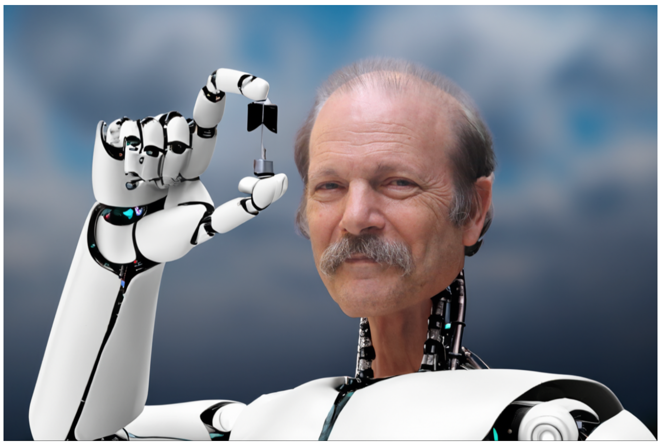
Please complete the following sentences?

- *Being a researcher* is what I enjoy and am good at.
  When solving a research problem, the community will be interested in your solution. Research is rewarding, but you need to be persistent.
- *My first research discovery* was different than the rest of all my other research results. It was on image processing, although it already used some kind of automated reasoning based on linear algebra.
- Being curious *is key to being a happy academic.*
- *Theoretical computer science in 100 years from now* will be even more important than today, solving even harder problems than the ones we face now.

Know the Person behind the Papers

Today: Moshe Vardi

Bio: *Moshe Y. Vardi is University Professor and the Karen Ostrum George Distinguished Service Professor in Computational Engineering at Rice University, where he is leading an Initiative on Technology, Culture, and Society. His interests focus on automated reasoning, a branch of Artificial Intelligence with broad applications to computer science, including machine learning, database theory, computational-complexity theory, knowledge in multi-agent systems, computer-aided verification, and teaching logic across the curriculum. He is also a Faculty Scholar at the Baker Institute for Public Policy at Rice University.*



EATCS: Nice to meet you, Moshe! This is actually the first live interview we conduct, usually the EATCS interviews are done offline.

MV: I very much prefer it live and interactive! This gives us more flexibility and you can ask additional questions spontaneously. This is also the reason why I don't teach with slides but on the board, "old school". This allows me to react to students better and also slows me down – a good control mechanism for my teaching speed.

EATCS: We ask all interviewees to share a photo with us.

MV: Most photos of me are made when the Rice University photographer shows up because he noticed that '*you look older than the picture on your website!*' I share two pictures. The first one was created by my son, for a poster for my lecture on robots and jobs. The second picture shows me with my wife, Pam, feeding a llama in Machu Pichu in 2018.

EATCS: Can you please tell us something about you that probably most of the readers of your papers don't know?

MV: I am a second-generation Holocaust survivor. I served 5 years in the military and was an artillery officer. This was a typical trajectory for people with a physics background like me.

EATCS: Is there a paper which influenced you particularly, and which you recommend other community members to read?

MV: One of my all-time favorite papers has a shortest possible title: *Alternation*, by Chandra, Kozen, and Stockmeyer. It is a very beautiful paper. I first read it as a paper on complexity theory but later it turned out that the concept of *alternating automata* (which were also introduced in that paper) became an important tool in my work on program verification. Alternation is really another way of game play, where players take turns: a major theme of my research. What was new in this paper is the idea to make games a computational construct. This idea is now also used intensively for synthesis. Alternation is also powerful because it is very close to logic. For example, the initial translation from LTL to Büchi automata is significantly simplified if taught or implemented via the concept of alternating automata.

EATCS: Is there a paper of your own you like to recommend the readers to study? What is the story behind this paper?

MV: My second most-cited paper is on the complexity of relational queries. I published this paper right after my PhD, but most of the research I conducted during my PhD, as a side project. It started with me being puzzled that there are seemingly contradictory complexity results for relational queries in the literature. One result claimed that the complexity of first-order queries is PSPACE complete, and another result claimed that the complexity of existential second-order queries is NP, which seemed strange as NP contains PSPACE but second-order logic is more expressive than first-order logic. So I realized that there are different ways to measure the complexity of queries, related to the expression complexity or the data complexity. Looking back, this was the first paper on what we call today multivariate complexity theory or parameterized complexity, studying how the different

input variables impact the complexity. I I sometimes also call it logical algorithmics, which I see as an attractive alternative way of developing algorithms, which fits database queries very well: rather than devising problem-specific algorithms one-by-one, there is a meta-algorithm and a to-be-studied property; the problem-specific algorithm is then obtained by compilation of this meta-algorithm. This idea came from relational database theory, which was the first topic of my research.

EATCS: Are there still open research challenges in this area today?

MV: Yes, for example in the context of constrained satisfaction problems. I am very interested to understand how we can use tree decompositions to evaluate queries faster. 20 years ago researchers were sceptic about this idea, as computing tree decompositions itself is a hard problem. However, now we not only have approximation algorithms but also good tools, and it is time to revisit these ideas, which may become tractable.

In general, I believe we need to revisit our definitions of tractability and complexity theory in general. Every year, tools like SAT solvers are getting significantly better. Problems that we thought are intractable, are now tractable in practice. We are now able to solve very large SAT instances that come from real-life applications.

What we thought is hard, is not hard. At the same time, what we thought is easy is not always easy: for several polynomial-time problems, we currently only have high-degree polynomial algorithms with huge constants.

We start to realize that complexity theory, one of the most beautiful theories we currently have, offers us only limited insights into the actual hardness of problems in practice. Ariel Rubinstein, the famous Israeli economist, wrote in his book on *Economic Fables* that our fundamental economical models can *only inform* economic decision making in the real world; the latter often comes with additional constraints and ultimately is also about values, politics, etc.

This is similar with complexity theory, which can provide us with guidelines and explanations, but not accurate predictions of practical run times.

EATCS: You are very productive and active on many fronts. Can you share with us how you manage your time? And when is your most productive working time?

MV: I am not a good time manager. I often juggle many balls, at any point of time several balls are on the floor. I just hope that the important balls on the floor will scream and catch my attention!

One has to learn to say no. At least I try to start with an empty inbox every year.

I am not a night owl, and I think sleep is very important, trying to be disciplined. One has to realize that a career is a marathon, not a sprint. I tried once, 30 years ago, to do an all-nighter to finish a paper, but I realized at midnight that my writing gets bad. So I never tried it again.

EATCS: Is there a nice anecdote from your career you like to share with our readers?

MV: When I was a postdoc, I had a single-author FOCS paper. At the reception of the conference, before my presentation, a colleague came to me and said that he really liked the paper. However, he got stuck in one of the proofs. I went to my room and noted a bug. I tried to fix it but I could not do it. So I had to give my presentation the next morning and say that a proof was wrong. Not a nice experience for a young postdoc! Fortunately, it turned out a few years later that my theorem was correct, but a new proof idea was necessary.

EATCS: Do you have any advice for young researchers? In what should they invest time, what should they avoid?

MV: People are different, and everyone has to figure out themselves what works for them. Also my students are very different, and I am not a good predictor of their success. I mainly judge students on their technical capabilities, which, however, are not necessary or sufficient for success. There are people who are technically not so strong but really good at managing their careers; and there are technically very strong researchers that don't act wisely. We are working within a social context, within a community, and we have to be aware of this. Think of Sheldon Cooper of Big Bang Theory, who is very smart but whose view of how the world works is very imperfect.

I have also seen very smart young researchers who take the dangerous strategy of focusing on solving a single very hard open problem. I always recommend to diversify the research. It is often difficult to guess upfront how hard a problem is going to be.

Generally, it is important to be able to create a vision and tell a story. I always ask my students to give a dry run for their talks, and try to deliver a story. All cultures love stories, and there is a universal 3-Act structure: introduce the characters, create a crisis for the characters, and resolve the crisis. I suggest to also use this structure for presenting your research.

EATCS: What are the most important features you look for when searching for graduate students?

MV: I don't search for students, they usually come to me. They typically took my class and I never declined an interested student. In contrast to Europe, in the U.S., the department selects the students and I usually try to work with my students as long as I can, and at least bring them to a masters degree.

EATCS: Do you see a main challenge or opportunity for theoretical computer scientists for the near future?

MV: The computer science field is still very young and evolving. For example, complexity theory is far from perfect and in a crisis. We need a new theory that lowers the discrepancy to the real world. I see the current situation as a huge opportunity: now we can become inventive again! We also had this big successes around machine learning, generative AI, and deep learning, which we don't understand well fundamentally. We need more theory to shed light on these approaches and why/when they work.

There are many success stories about theories which became very relevant in practice. One of my most successful research results, related to automata-theoretic model checking, actually built upon fundamental results that were originally derived in purely theoretical studies, without specific applications in mind. The elegant theories developed in the context of decision procedures for monadic structures or fixpoints, turned out to have very practical applications and gave rise to industrial tools.

There is also this debate about beauty. For example, physicists consider string theory beautiful, but some believe that it led us astray. Beauty does not always lead to usefulness. But in case of automata-theoretic model checking, the simplicity and beauty was actually important to make it practically relevant, as it is easy to understand and implement. I believe that it is not about beauty but about simplicity, for implementation.

I believe that good theory has huge potential. One of the success stories in computer science, relational databases, relies on query languages based on first-order logic theory. Initially, systems researchers were very concerned that this approach will never perform well and hence not be practical. In fact, there were big controversies at that time, for example inside IBM. And now declarative query languages are the foundations of our Western Civilization. It was one of the research contributions that received the Turing Award the fastest.

EATCS: Can you recommend some source of information that you enjoy (e.g., a specific blog, podcast, youtube channel, book, show, ...)?

MV: One of the books that I like very much is *The Universal Computer: The Road from Leibniz to Turing*, by Martin Davis. It tells the story how computing arose, going back all the way to Leibniz. Usually we teach the students mainly how the algorithms and theories work, but not much about the history how these concepts were developed. I think it is a mistake not to include the historical dimension. All these great ideas were created by human beings, and students should understand also this dimensions.

Actually I also gave a similar talk, *From Aristotle to the iPhone* (`https://www.youtube.com/watch?v=iWWqUIzDIeQ`), in which I try to tell this story in one hour. I gave this talk in different versions. Once I was asked to give it at UCLA, in a more lively version. So I added one line to the abstract: *this is a story where the protagonists usually die young, miserably, or both.* And this attracted a big audience, and I realized how much people are interested in the human dimension. I also gave this talk at the UNESCO Logic Day. The interest was so large that our Zoom licence had to be extended multiple times to accommodate about one thousand listeners.

Please complete the following sentences?

- *Being a researcher...* I tell you a story. When I was in the army, we had to practice parachute jumps. Once I was last to get onto the plane, and so I had to be first at the door from where the jump takes place. Standing there for minutes in front of the open air, engines roaring, ready to jump. These moments between plane and the "nothingness", is a most alive and exciting feeling. This is a beautiful metaphor for research: we are standing at the edge of the unknown, and all we have to do is jump, into an exciting space. I want my students to feel the same.

- *My first research discovery...* was actually accidental. I was a master student and did not know what to explore. I attended some seminar and I got to present a paper, which at the end raised an open question. This problem intrigued me and this became my master thesis. And then this topic even evolved into my PhD thesis.

- Being resilient ... *is key to being a happy academic.* Being professor is not an easy job. People get their PhDs at the end of their 20ties, and then need to be productive researchers for 40 years. Topics evolve significantly over these time frames, and even if you have an excellent reputation, it stays competitive and your papers still get rejected. In fact, sometimes the expectations on you are even higher if you are more successful. You also need to be able to cope with the freedom that one has.

- *Theoretical computer science in 100 years from now...* It is hard to give predictions, as the field is still young, compared to other disciplines. It still needs to mature. For example I am very curious how complexity theory will evolve. Will it be a different theory? Or will the current one be refined? My hope is that we will have theories that give better answers. I think the best is yet to come!

The Education Column

by

Juraj Hromkovič and Dennis Komm

ETH Zürich, Switzerland

`juraj.hromkovic@inf.ethz.ch` and `dennis.komm@inf.ethz.ch`

Teaching Formal Foundations of Computer Science with Iltis

Marko Schmellenkamp
Ruhr University Bochum
marko.schmellenkamp@rub.de

Fabian Vehlken
Ruhr University Bochum
fabian.vehlken@rub.de

Thomas Zeume
Ruhr University Bochum
thomas.zeume@rub.de

Abstract

Introductory courses on formal foundations of computer science are often attended by large numbers of students with diverse backgrounds. In this paper we outline how we address this challenge in our courses by supplementing traditional teaching with web-based, interactive exercises. The web-based exercises are provided by Iltis, a modern teaching support system covering the foundations of computer science logic, formal languages, and (parts of) complexity theory. We give a gentle introduction to Iltis, describe its technical integration into our courses, and outline research challenges and opportunities coming up when developing such a system.

1 Introduction and Motivation

Formal foundations are at the core of many modern applications of computer science and are therefore an integral part in recommendations for Bachelor computer science curricula [8, 6]. Typical study programs implement these recommendations by offering, among others, courses on *Logics for Computer Scientists* — covering the reasoning pipeline for propositional logic and first-order logic — and on *Theoretical Foundations of Computer Science* — covering formal languages as well as basics of complexity and computability theory; see Figure 1(a) for an overview of standard topics.

Teaching these formal foundations is a challenge for most instructors as it is one of the harder topics for students. Also, increasing numbers of students enrolled in computer science courses with diverse backgrounds are difficult to

Foundations of logic

- Basics: Propositional, modal, and first-order logic
- Advanced: Logics for verification, description logics, etc.
- Methods: Modelling, reasoning pipeline (modelling, transformation, inference), algorithms for evaluation & satisfiability

Foundations of formal languages

- Basics: Regular and context-free languages
- Advanced: Regular tree languages, timed languages etc.
- Methods: Modelling, closure properties, constructions & algorithms, pumping lemmata

Basics of complexity theory

- Basics: P, NP, completeness
- Advanced: Space-based classes, fine-grained complexity, etc.
- Methods: Classifications, closure properties, reductions, . . .

Basics of computability theory

- Basics: (Semi-)decidability, undecidability, computability
- Methods: Classifications, closure properties, reductions, . . .

(a)

Exercise: From modelling to inference

Julia has identified the following dependencies between program libraries and system libraries:

- No software package is both a program library and a system library.
- System libraries depend only on system libraries.
- Every software package that must be explicitly installed by the user depends on at least one program library directly.

She concludes that there is no system library that must be explicitly installed by the user.

Can you confirm her conclusion using methods you learned for first-order logic?

(b)

Figure 1: (a) Topics typically covered by courses on formal foundations of computer science. (b) A typical exercise for a workflow covering modelling, transformation, and inference in first-order logic.

handle with traditional lecture- and tutorial-based courses. In particular, providing individual human tutoring for such a large number of students with diverse needs exceeds the resources of most CS departments. A general approach for tackling this challenge and increasing learning outcomes across STEM disciplines is provided by the National Research Council of the US which advocates, among others, to "Leverage technologies to make the most effective use of students' time, shifting from information delivery to sense-making and practice in class" [9, 2]. For formal foundations, technological teaching support in particular may help to make room for theory and in-depth problem solving in lectures and tutorials by outsourcing some basics.

From our perspective, to be useful in large, mandatory courses, teaching support technologies for formal foundations of computer science need to offer:

- *Coverage* of a wide range of topics in formal foundations of computer science;
- *Advanced feedback and support* provided immediately, extensively, and individually;
- *Flexibility* in how to use and combine educational tasks;
- *Easy integration* into courses; and
- *Extensibility* of topical range and feedback mechanisms.

Many teaching support systems for topics typically taught in introductory formal foundations courses have been developed over the years. Most of these systems were developed ad-hoc by instructors for helping their students. A common theme is that only a small set of topics (typically only one) is covered; systems are abandoned and/or become technologically outdated rather quickly; and in most of them only very basic feedback is provided.

In this article we report on our experiences and progress in building the teaching support system Iltis.[1] In short, Iltis offers a wide range of interactive, web-based exercises on formal foundations of computer science. It is designed for flexibility and extensibility, and offers easy integration into common learning management systems. Within this article we address

- the scope of Iltis – which topics are covered and how content can be composed for different needs (see Section 2);
- how we set up large introductory courses on *Logic for Computer Science* and on *Foundations of Theoretical Computer Science* with integrated web-based exercises in Iltis (see Section 3);
- what research challenges and opportunities arise – theoretical, practical, and didactical – when building teaching support systems for formal foundations of computer science (see Section 4).

This article updates, adapts, and condenses a report from 2021 by a superset of the current authors [5].

2 An Introduction to Iltis

In Iltis, instructors can design educational content flexibly by using a broad portfolio of educational tasks in foundations of logic, formal languages, and complexity theory (see Section 2.3). A compositional task model allows to combine tasks flexibly into multi-step exercises (see Section 2.1). A compositional feedback model allows for providing feedback according to the progress of students in curricula (see Section 2.2).

2.1 Compositional Task Model

Exercises in Iltis are built from small, easily composable, educational tasks. Each educational task is configurable by inputs — either given explicitly or as the output of prior tasks — and provides objects created by students within this task as outputs. The outputs can then be used by subsequent educational tasks. For instance, a

[1]Iltis [ˈɪltɪs] is the German word for polecat and the Swiss animal of the year 2024 [1]. We invite all readers to try out Iltis: `https://iltis.cs.tu-dortmund.de/`

Table 1: A summary of educational tasks in the logic domain that are supported by Iltis.

Task	Propositional logic	Modal logic	First-order logic
Evaluating formulas	✓	✓	–
Constructing models	✓	✓	✓
Creating signatures	✓	✓	(✓)
Constructing formulas	✓	✓	✓
Transforming	✓	✓	✓
Testing satisfiability	✓	✓	✓
Task variants & further tasks	Satisfiability tests with • truth tables • HornSat algorithm • tableau calculus • resolution	Satisfiability test with tableau calculus Calculating bisimulations Proving non-bisimilarity of worlds	Satisfiability test with resolution Proving non-equivalence of formulas

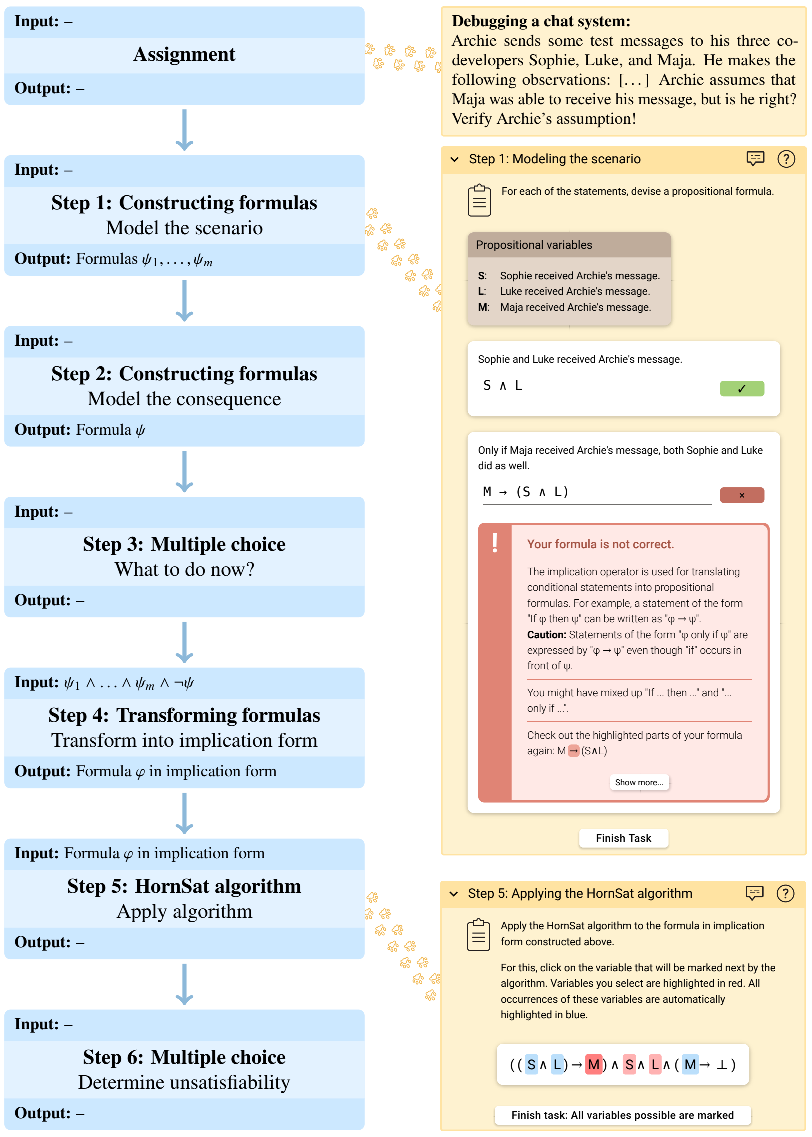
Table 2: A summary of educational tasks that are supported for formal languages, computability and complexity theory.

Regular languages	Context-free languages	Computability & complexity theory
Modeling with • deterministic automata • non-deterministic automata • regular expressions	Modeling with • push-down automata • deterministic push-down automata • context-free grammars (CFGs)	Interacting with graphs: • constructing graphs • colouring nodes and edges satisfying multiple conditions
Specifying words Specifying Myhill-Nerode-classes Proving non-equivalence of languages	Specifying words Specifying derivations in CFGs Proving non-equivalence of languages	Specifying graph reductions

task for transforming a formula into conjunctive normal form (CNF) receives a formula as input and provides the student-constructed, equivalent formula in CNF as output.

Typical workflows used in formal foundations of computer science can be covered by multi-step exercises composed of different educational tasks. For instance, Figure 2 illustrates a workflow in which students first model a scenario with propositional formulas φ_1 and φ_2, and then infer another propositional formula ψ by first deciding what to do, then transforming $\varphi_1 \land \varphi_2 \land \neg\psi$ into CNF, and finally showing unsatisfiability of the set of clauses via the satisfiability algorithm for Horn formulas (instead of the latter, also propositional resolution or the propositional tableau calculus could be used). Throughout this workflow, the formulas entered by the students are used for subsequent tasks.

Two further multi-step exercises are illustrated in Figures 2, 3 and 5.


Input: –
Assignment
Output: –
Input: –
Step 1: Constructing formulas
Model the scenario
Output: Formulas $\psi_1, \ldots, \psi_m$
Input: –
Step 2: Constructing formulas
Model the consequence
Output: Formula ψ
Input: –
Step 3: Multiple choice
What to do now?
Output: –
Input: $\psi_1 \wedge \ldots \wedge \psi_m \wedge \neg\psi$
Step 4: Transforming formulas
Transform into implication form
Output: Formula φ in implication form
Input: Formula φ in implication form
Step 5: HornSat algorithm
Apply algorithm
Output: –
Input: –
Step 6: Multiple choice
Determine unsatisfiability
Output: –
Step 1: Modeling the scenario
Finish Task
Step 5: Applying the HornSat algorithm
Show more...
Finish task: All variables possible are marked

Debugging a chat system:
Archie sends some test messages to his three co-developers Sophie, Luke, and Maja. He makes the following observations: [. . .] Archie assumes that Maja was able to receive his message, but is he right? Verify Archie's assumption!

For each of the statements, devise a propositional formula.

Propositional variables

S:	Sophie received Archie's message.
L:	Luke received Archie's message.
M:	Maja received Archie's message.

Sophie and Luke received Archie's message.

`S ∧ L` ✓

Only if Maja received Archie's message, both Sophie and Luke did as well.

`M → (S ∧ L)` ×

Your formula is not correct.

The implication operator is used for translating conditional statements into propositional formulas. For example, a statement of the form "If φ then ψ" can be written as "φ → ψ". **Caution:** Statements of the form "φ only if ψ" are expressed by "φ → ψ" even though "if" occurs in front of ψ.

You might have mixed up "If ... then ..." and "... only if ...".

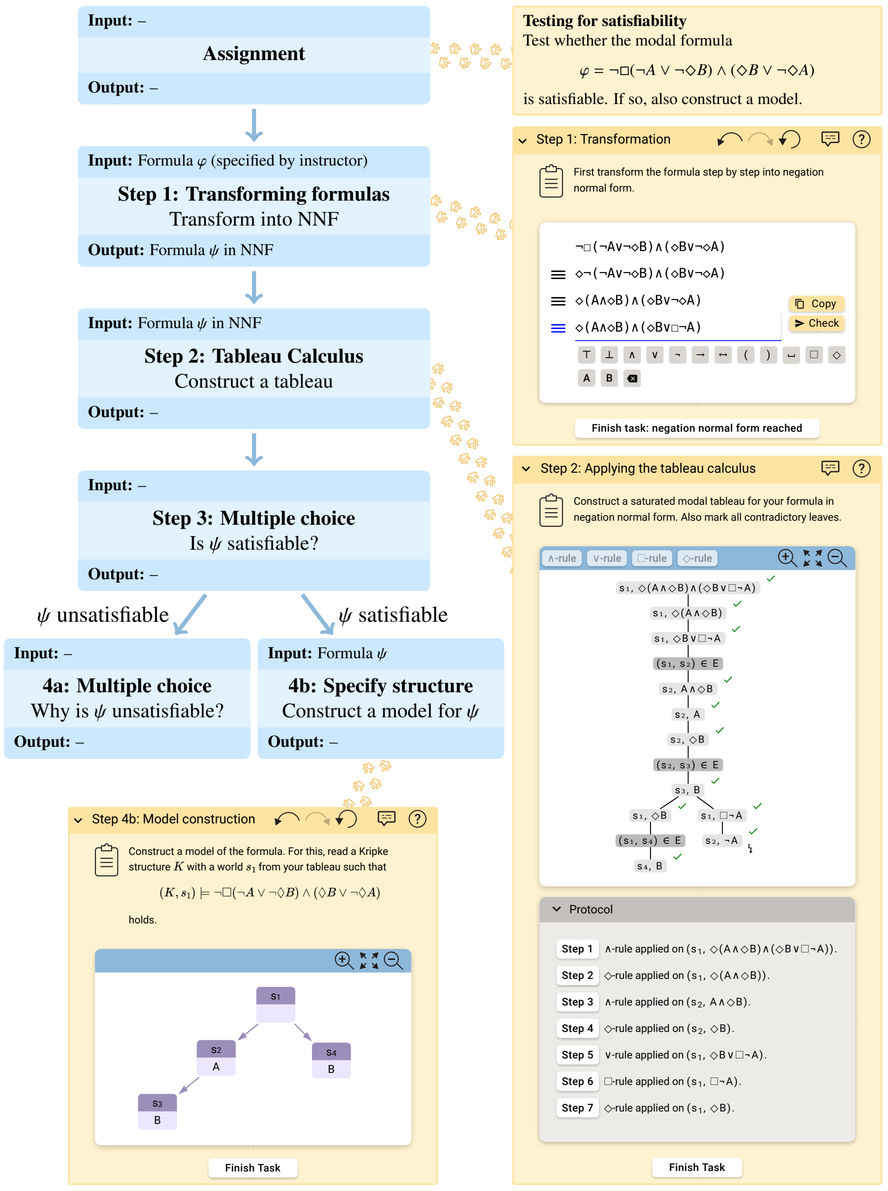
Check out the highlighted parts of your formula again: M → (S∧L)

Apply the HornSat algorithm to the formula in implication form constructed above.

For this, click on the variable that will be marked next by the algorithm. Variables you select are highlighted in red. All occurrences of these variables are automatically highlighted in blue.

$((S \wedge L) \rightarrow M) \wedge S \wedge L \wedge (M \rightarrow \bot)$

Figure 2: An exercise for the propositional reasoning workflow, composed of smaller educational tasks. For this sample scenario, the instructor chose the HornSat satisfiability test as Horn formulas are sufficiently expressive. For general propositional formulas, also truth tables, propositional resolution, and the propositional tableau calculus can be used. In Step 1, the student chose to reveal the first three feedbacks.



Input: –
Assignment
Output: –

Input: Formula φ (specified by instructor)
Step 1: Transforming formulas
Transform into NNF
Output: Formula ψ in NNF

Input: Formula ψ in NNF
Step 2: Tableau Calculus
Construct a tableau
Output: –

Input: –
Step 3: Multiple choice
Is ψ satisfiable?
Output: –

ψ unsatisfiable

Input: –
4a: Multiple choice
Why is ψ unsatisfiable?
Output: –

ψ satisfiable

Input: Formula ψ
4b: Specify structure
Construct a model for ψ
Output: –

Testing for satisfiability
Test whether the modal formula

$$\varphi = \neg\Box(\neg A \vee \neg\Diamond B) \wedge (\Diamond B \vee \neg\Diamond A)$$

is satisfiable. If so, also construct a model.

Step 1: Transformation

First transform the formula step by step into negation normal form.

```
  ¬□(¬A∨¬◇B)∧(◇B∨¬◇A)
≡ ◇¬(¬A∨¬◇B)∧(◇B∨¬◇A)
≡ ◇(A∧◇B)∧(◇B∨¬◇A)
≡ ◇(A∧◇B)∧(◇B∨□¬A)
```

Step 2: Applying the tableau calculus

Construct a saturated modal tableau for your formula in negation normal form. Also mark all contradictory leaves.

Protocol

Step 1	∧-rule applied on $(s_1, \Diamond(A\wedge\Diamond B)\wedge(\Diamond B\vee\Box\neg A))$.
Step 2	◇-rule applied on $(s_1, \Diamond(A\wedge\Diamond B))$.
Step 3	∧-rule applied on $(s_2, A\wedge\Diamond B)$.
Step 4	◇-rule applied on $(s_2, \Diamond B)$.
Step 5	∨-rule applied on $(s_1, \Diamond B\vee\Box\neg A)$.
Step 6	□-rule applied on $(s_1, \Box\neg A)$.
Step 7	◇-rule applied on $(s_1, \Diamond B)$.

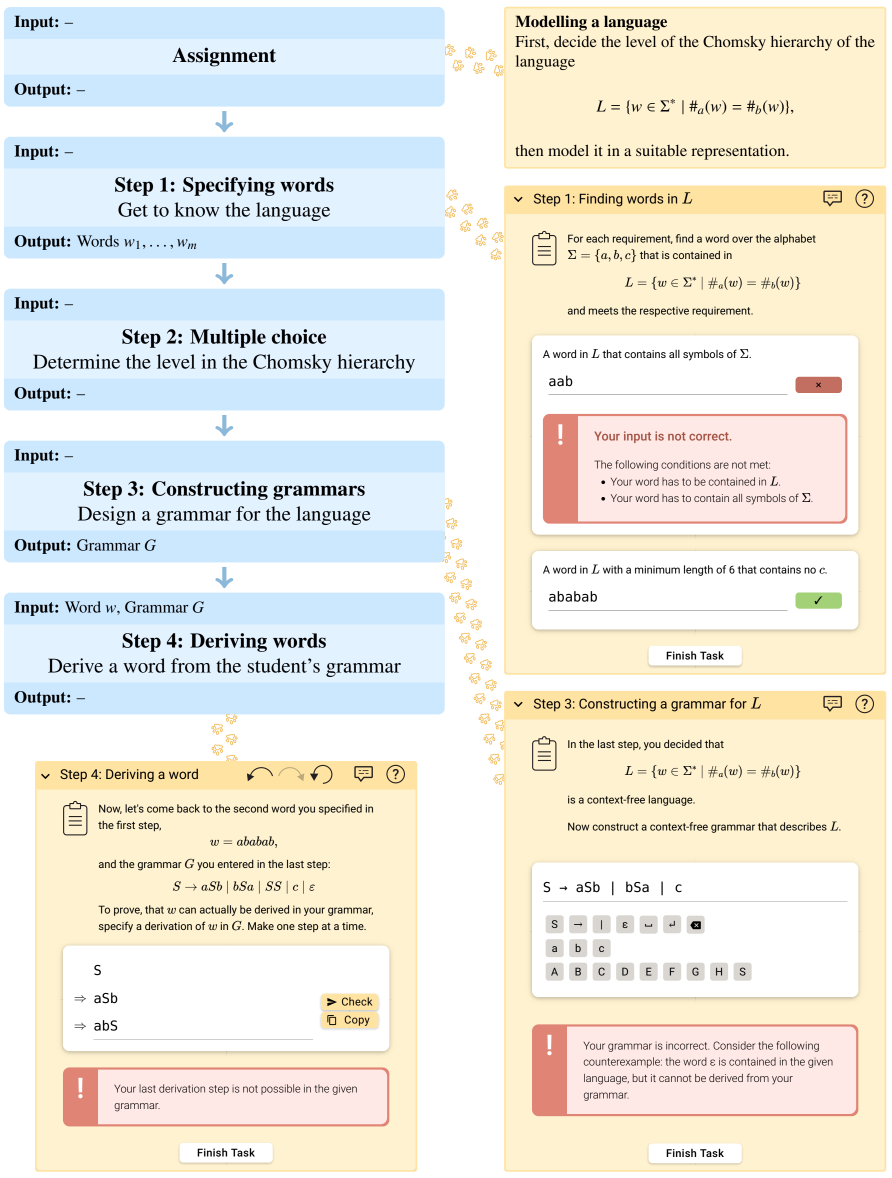
Step 4b: Model construction

Construct a model of the formula. For this, read a Kripke structure K with a world s_1 from your tableau such that

$$(K, s_1) \models \neg\Box(\neg A \vee \neg\Diamond B) \wedge (\Diamond B \vee \neg\Diamond A)$$

holds.

Figure 3: An exercise for solving the satisfiability problem for modal formulas, composed of smaller educational tasks. For satisfiable and unsatisfiable formulas, different workflows can be used.



Modelling a language
First, decide the level of the Chomsky hierarchy of the language

$$L = \{w \in \Sigma^* \mid \#_a(w) = \#_b(w)\},$$

then model it in a suitable representation.

Step 1: Finding words in L

For each requirement, find a word over the alphabet $\Sigma = \{a, b, c\}$ that is contained in

$$L = \{w \in \Sigma^* \mid \#_a(w) = \#_b(w)\}$$

and meets the respective requirement.

A word in L that contains all symbols of Σ.

aab ×

Your input is not correct.

The following conditions are not met:

- Your word has to be contained in L.
- Your word has to contain all symbols of Σ.

A word in L with a minimum length of 6 that contains no c.

ababab ✓

Step 3: Constructing a grammar for L

In the last step, you decided that

$$L = \{w \in \Sigma^* \mid \#_a(w) = \#_b(w)\}$$

is a context-free language.

Now construct a context-free grammar that describes L.

```
S → aSb | bSa | c
```

Your grammar is incorrect. Consider the following counterexample: the word ε is contained in the given language, but it cannot be derived from your grammar.

Step 4: Deriving a word

Now, let's come back to the second word you specified in the first step,

$$w = ababab,$$

and the grammar G you entered in the last step:

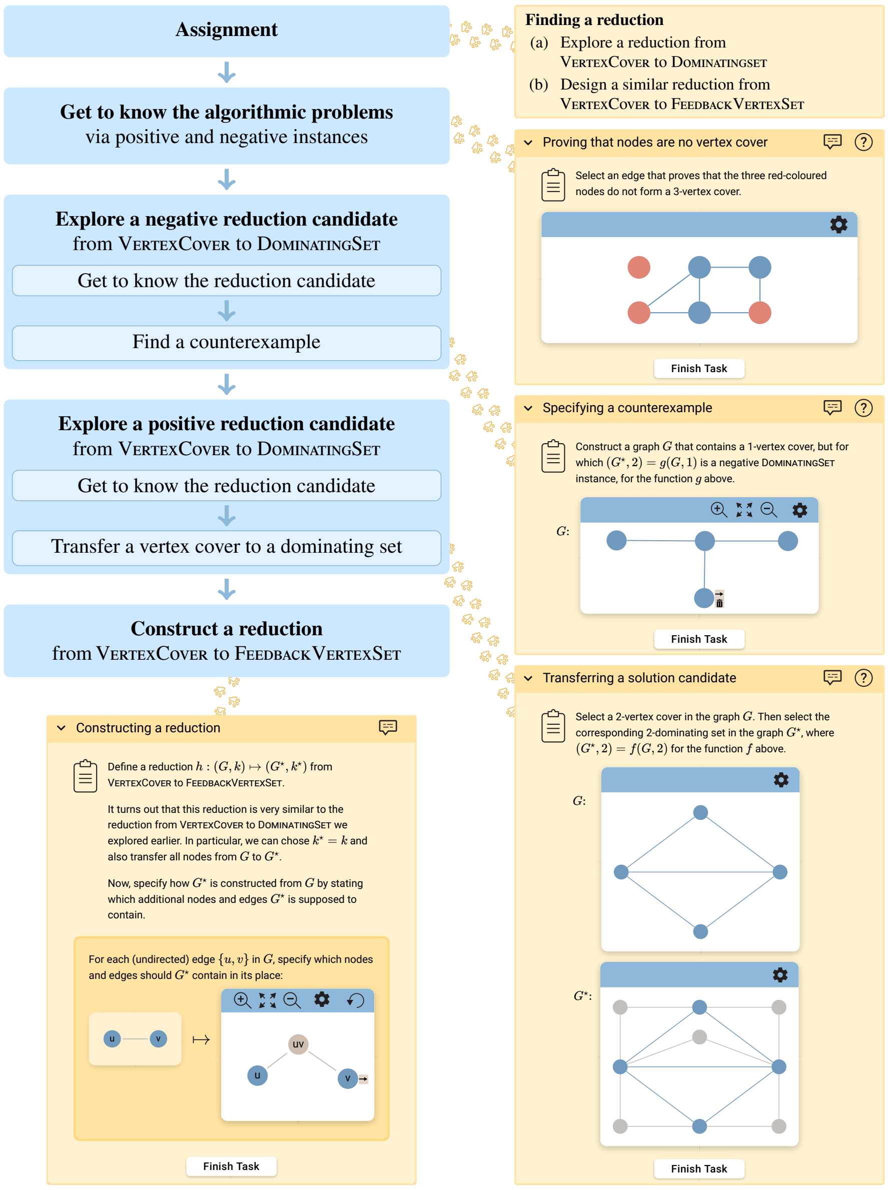
$$S \to aSb \mid bSa \mid SS \mid c \mid \varepsilon$$

To prove, that w can actually be derived in your grammar, specify a derivation of w in G. Make one step at a time.

```
  S
⇒ aSb
⇒ abS
```

Your last derivation step is not possible in the given grammar.

Figure 4: An exercise for constructing a representation of a formal language. As preparatory step, students explore the formal language by identifying some elements and deciding its level in the Chomsky hierarchy. For constructing a representation of a context-free languages Iltis supports context-free grammars and push-down automata; for regular languages, it supports (non-)deterministic finite state automata and regular expressions.


Assignment
Get to know the algorithmic problems
via positive and negative instances
Explore a negative reduction candidate
from VertexCover to DominatingSet
Get to know the reduction candidate
Find a counterexample
Explore a positive reduction candidate
from VertexCover to DominatingSet
Get to know the reduction candidate
Transfer a vertex cover to a dominating set
Construct a reduction
from VertexCover to FeedbackVertexSet
Proving that nodes are no vertex cover
Finish Task
Specifying a counterexample
G:
Finish Task
Transferring a solution candidate
G:
G*:
Finish Task
Constructing a reduction
u
v
↦
uv
u
v
Finish Task

Finding a reduction

(a) Explore a reduction from VertexCover to DominatingSet

(b) Design a similar reduction from VertexCover to FeedbackVertexSet

Select an edge that proves that the three red-coloured nodes do not form a 3-vertex cover.

Construct a graph G that contains a 1-vertex cover, but for which $(G^*, 2) = g(G, 1)$ is a negative DominatingSet instance, for the function g above.

Select a 2-vertex cover in the graph G. Then select the corresponding 2-dominating set in the graph G^*, where $(G^*, 2) = f(G, 2)$ for the function f above.

Define a reduction $h : (G, k) \mapsto (G^*, k^*)$ from VertexCover to FeedbackVertexSet.

It turns out that this reduction is very similar to the reduction from VertexCover to DominatingSet we explored earlier. In particular, we can chose $k^* = k$ and also transfer all nodes from G to G^*.

Now, specify how G^* is constructed from G by stating which additional nodes and edges G^* is supposed to contain.

For each (undirected) edge $\{u, v\}$ in G, specify which nodes and edges should G^* contain in its place:

Figure 5: A series of tasks for helping students to find a computational reduction between two problems. As a preparatory step, students can explore (a) the involved algorithmic problems and (b) reduction candidates similar to the reduction to be found (one incorrect and one correct candidate). The actual workflow is only depicted partially in this illustration.

2.2 Compositional Feedback Model

One of the core objectives of Iltis is to provide immediate and comprehensive feedback, as this is one of the most important factors for learning success. Educational task types in Iltis come with multiple *feedback generators*, each one responsible for one kind of feedback. Individual feedback generators can be composed to *feedback strategies* by simple rule-based programs. Such programs are executed upon student input and determine the order of feedback, with the execution of rules possibly depending on the result of previous rules.

When specifying interactive exercises, instructors can state which feedback strategy to use (or they can define a custom one). In this way, the progress of students can be taken into account, e.g., a strategy that provides a lot of feedback can be used for beginners, while a strategy that provides almost no feedback can be used for exam preparation. By gradually uncovering the feedback from the different generators, students can choose how much of the feedback provided they want to use.

Designing feedback strategies and generators is a subtle and challenging task as algorithmic feasibility as well as didactical aspects have to be taken into account. We sketch a sample strategy and its feedback generators for providing feedback for the construction of propositional formulas (see Figure 2, Step 1, for a partial illustration):

(1) *Correctness:* Is the constructed formula *correct* or *not correct*?

(2) *Misconceptions:* Typical misconceptions (e.g., mixing up premise and conclusion of an implication, especially when modelling "only if"-statements) are identified using an abstract rule framework [4]. They are the basis for several feedback generators:

  (a) *Hint at the misconception* (e.g., "Do you remember how 'if'- and 'only if'-statements can be expressed in propositional logic?")

  (b) *State the misconception explicitly* (e.g., "You might have mixed up 'if' and 'only if'.")

  (c) *Point out the precise position of the mistake*

(3) *Distinguishing model:* A valuation that distinguishes the constructed formula from a correct formula.

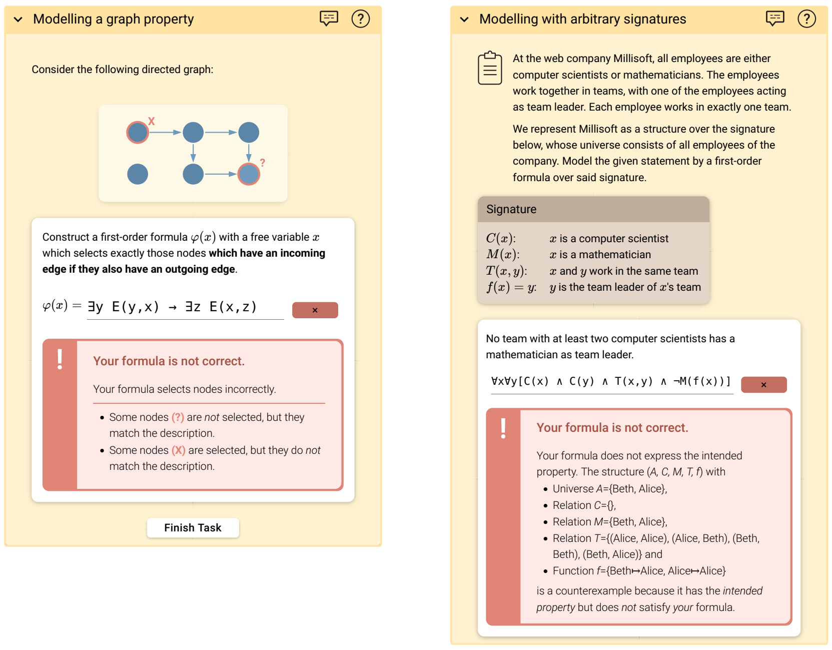
2.3 Educational Tasks

Iltis supports a variety of educational tasks for foundations of computer science logic, formal languages, and (parts of) complexity theory. In addition to content-specific tasks, there are also tasks to smooth out multi-step exercises, such as multiple-choice tasks.

Foundations of logic. Educational tasks for foundations of logic cover propositional logic, modal logic, and first-order logic content (see Table 1 for an overview). A broad spectrum of typical tasks is covered, including:

- *Evaluating formulas:* Students can evaluate formulas for a given interpretation by constructing truth tables for propositional formulas or evaluation tables for a given modal formula and Kripke structure, respectively.
- *Constructing models:* Students can construct models (i.e., satisfying interpretations) for formulas. For propositional logic, models are specified by valuations for all variables, for modal logic by Kripke structures; and for first-order logic, students can construct structures.
- *Creating signatures:* Students can specify a suitable logical language for describing a scenario. For propositional and modal logic, students can specify which propositional variables they want to use and describe their meaning in natural language (see Figure 7). For first-order logic, this is currently being implemented. A prototype for describing first-order signatures in natural language is available.
- *Constructing formulas:* Students can construct formulas for natural-language descriptions of scenarios provided by the instructor. For first-order logic, there is a second variant where teachers can provide a textual description of a unary graph query as well as a sample graph, and students are asked to provide a first-order formula that selects the same nodes as the graph query on this sample graph (see Figure 6 for both variants).
- *Applying equivalence transformations:* Students can transform propositional, modal, and first-order formulas step-by-step either into an equivalent target formula or into an equivalent formula in a given normal form.
- *Testing satisfiability:* Students can test formulas for satisfiability using several methods such as resolution or the tableau calculus.

Foundations of formal languages. Educational tasks for foundations of formal languages cover regular languages and context-free languages (see Table 2 for an overview):



Modelling a graph property

Consider the following directed graph:

Construct a first-order formula $\varphi(x)$ with a free variable x which selects exactly those nodes **which have an incoming edge if they also have an outgoing edge**.

$\varphi(x) =$ `∃y E(y,x) → ∃z E(x,z)`

Your formula is not correct.

Your formula selects nodes incorrectly.

- Some nodes (?) are *not* selected, but they match the description.
- Some nodes (X) are selected, but they do *not* match the description.

Finish Task

Modelling with arbitrary signatures

At the web company Millisoft, all employees are either computer scientists or mathematicians. The employees work together in teams, with one of the employees acting as team leader. Each employee works in exactly one team.

We represent Millisoft as a structure over the signature below, whose universe consists of all employees of the company. Model the given statement by a first-order formula over said signature.

Signature

$C(x)$:	x is a computer scientist
$M(x)$:	x is a mathematician
$T(x,y)$:	x and y work in the same team
$f(x) = y$:	y is the team leader of x's team

No team with at least two computer scientists has a mathematician as team leader.

`∀x∀y[C(x) ∧ C(y) ∧ T(x,y) ∧ ¬M(f(x))]`

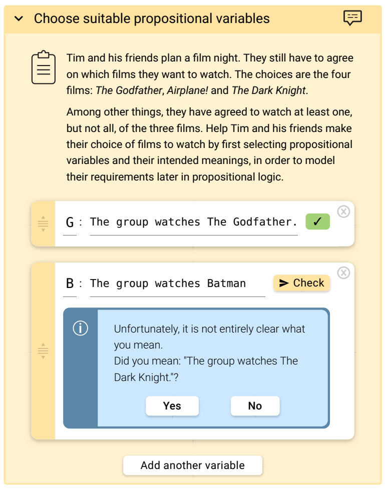
Your formula is not correct.

Your formula does not express the intended property. The structure (*A, C, M, T, f*) with

- Universe *A*={Beth, Alice},
- Relation *C*={},
- Relation *M*={Beth, Alice},
- Relation *T*={(Alice, Alice), (Alice, Beth), (Beth, Beth), (Beth, Alice)} and
- Function *f*={Beth↦Alice, Alice↦Alice}

is a counterexample because it has the *intended property* but does *not* satisfy *your* formula.

Figure 6: Educational tasks for constructing first-order formulas over graph signatures (left) and general signatures (right).



Choose suitable propositional variables

Tim and his friends plan a film night. They still have to agree on which films they want to watch. The choices are the four films: *The Godfather*, *Airplane!* and *The Dark Knight*.

Among other things, they have agreed to watch at least one, but not all, of the three films. Help Tim and his friends make their choice of films to watch by first selecting propositional variables and their intended meanings, in order to model their requirements later in propositional logic.

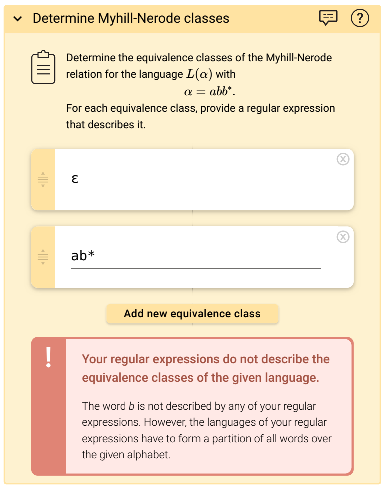
`G : The group watches The Godfather.` ✓

`B : The group watches Batman` Check

Unfortunately, it is not entirely clear what you mean.
Did you mean: "The group watches The Dark Knight."?

Yes No

Add another variable



Determine Myhill-Nerode classes

Determine the equivalence classes of the Myhill-Nerode relation for the language $L(\alpha)$ with $\alpha = abb^*$.
For each equivalence class, provide a regular expression that describes it.

`ε`

`ab*`

Add new equivalence class

Your regular expressions do not describe the equivalence classes of the given language.

The word *b* is not described by any of your regular expressions. However, the languages of your regular expressions have to form a partition of all words over the given alphabet.

Figure 7: Educational task for specifying propositional variables and their intended meaning. The intended meaning is verified via natural language processing models (currently fine-tuned for German exercises).

Figure 8: Educational task for identifying the Myhill-Nerode classes of a language.

- *Modelling:* Students can model languages with a variety of representations. For regular languages, (non-)deterministic automata and regular expressions can be used. For context-free languages, deterministic and general push-down automata and context-free grammars can be used.
- *Specifying Myhill-Nerode classes:* Students can specify regular expressions for the equivalence classes of the Myhill-Nerode relation.
- *Specifying derivations:* Students can derive a given word from a given context-free grammar step-by-step. For each step, Iltis checks whether it is valid.
- *Specifying words:* Students can specify words over a given alphabet. Then, Iltis checks whether they are contained in given (combinations of) regular or context-free languages. In this way, students can also prove the non-equivalence of languages.

Foundations of complexity theory. Educational tasks for foundations of complexity theory and computability theory are currently under development. In complexity theory, educational tasks for understanding algorithmic problems and computational reductions are already covered:

- *Interacting with graphs:* For understanding graph problems, students can select and colour nodes and edges in given graphs and build new graphs from scratch. The user input can be tested for a variety of conditions. This educational task can be used very flexibly.
- *Specifying graph reductions:* Students can specify certain graph reductions by specifying in a modular way how to map nodes and edges from an instance of the source problem to an instance of the target problem.

3 Instructor's Perspective: A Course Set-Up

We integrated web-based exercises provided by Iltis into our courses *Logic in Computer Science* (Bachelor, mandatory, 2 + 1 contact hours)[2] and *Foundations of Theoretical Computer Science* (Bachelor, mandatory, 4 + 2 contact hours)[3] at Ruhr University Bochum, each with > 200 students. The exercises are also used in similar courses at TU Dortmund University with > 400 students.

[2]The web-based exercises for *Logic in Computer Science* can be found at `https://iltis.cs.tu-dortmund.de/Logic-external/de/`

[3]The web-based exercises for *Foundations of Theoretical Computer Science* can be found at `https://iltis.cs.tu-dortmund.de/TCS-external/de/`

Course organisation. The set-up of the courses differs slightly, we focus on the *Foundations of Theoretical Computer Science* course covering regular languages, context-free languages, an introduction to computability theory, and an introduction to complexity theory. Organisation-wise, the course consists of:

- *Lectures:* Two traditional 90-minute lectures per week with all students, interrupted by few questions testing understanding via a student classroom response system.
- *Tutorials:* One 90-minute tutorial per week in groups of 20–30 students. First half spent on active problem solving in small groups and discussion. Second half spent on discussing solutions to assignments.
- *Assignments:* Consisting of (a) interactive web-based exercises provided by Iltis, some graded and some ungraded; (b) traditional assignments graded by tutors. The interactive web-based exercises are typically easier and students can try as often as they want, receiving feedback from Iltis for each attempt.

Our objective for the integration of interactive, web-based exercises was two-fold. First, to offer students the opportunity to train basics and receive feedback very early on. Having understood the basics, they then go on to tutorials and to the more complex traditional assignments. Second, outsourcing the basics to Iltis helps to save valuable time of teaching assistants, which then can be used to discuss more difficult topics and for in-depth problem solving in tutorials. The web-based exercises are provided at the time of the lecture and our recommendation for students is to do the web-based exercises before going to tutorials and starting with the analog exercises.

Technical organisation. The assignments – web-based and analog – are managed through our universities learning management platform Moodle. Both web-based and analog exercises have a digital twin in Moodle. The grading of the web-based exercises is handled by Iltis; the grading of the analog exercises by teaching assistants. The accounting of points is handled by Moodle.

Iltis exercises are integrated into the learning management platform via the LTI standard [10], which is supported by many modern teaching management platforms. They are stored in easily configurable XML files which are managed via Git repositories. Content for new courses can be created by starting from a clone of an existing course, selecting from a portfolio of existing exercises and adapting them according to the requirements.

4 Research Challenges & Opportunities

Building teaching support systems for formal foundations of computer science comes with a multitude of challenges and research opportunities. We sketch some of them.

A theory challenge

Providing teaching support for formal foundations of computer science – including feedback and advice for students and learning analytics for instructors – requires to solve algorithmic problems that are in most cases provably algorithmically hard or even unsolvable.

The main road block for teaching support systems for formal foundations is that many of the algorithmic tasks that need to be solved are inherently hard or even algorithmically unsolvable in general. For instance, deciding whether a scenario has been correctly modelled by a first-order formula is algorithmically impossible in general, due to the undecidability of testing whether two first-order formulas have the same meaning. While this is possible for propositional logic, no efficient algorithm is known so far. The same algorithmic hardness holds for many educational tasks for the foundations of logics, formal languages, complexity theory, and computability theory. Providing feedback beyond the mere fact whether a solution is correct – i.e., feedback that hints at the students' misconceptions or gently guides students to correct solutions – is potentially even harder.[4]

Attacking this challenge requires to find creative approaches for circumventing algorithmic hardnesses and often leads to interesting theoretical research questions. Hope for successfully tackling this challenge is provided by the fact that human tutors manage to provide feedback and advice.

We sketch two concrete examples, where methods from theoretical computer science helped us to come up with approaches for providing meaningful feedback and advice for students:

- **Explaining mistakes in context-free grammars.** In ongoing work on providing explanations for mistakes in modelling with context-free formalisms, we aim for feedback along the lines of the following interaction.

  Assignment: *Design a context-free grammar for* $L = \{a^n b^{n+2} \mid n \in \mathbb{N}\}$.

  Student: $S \to aSb \mid abb$

  Feedback (provided in customizable stages):

  (1) Correctness: *Your grammar is not correct.*

[4]The web-based teaching support system *AutomataTutor* provides advanced, high-level feedback for finite automata constructed by students [3].

(2) Pinpointing mistakes:

(a) Hinting at a wrong language: *Your grammar describes the language*

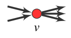
$$L = \{a^n b^{n+1} \mid n \in \mathbb{N}\}.$$

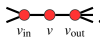
(b) Hinting at a wrong rule: *It might help to have a look at the rule*

$$S \to abb.$$

(c) Provide a counterexample: *The language described by your grammar contains the word abb, which is not in L.*

Even though testing correctness of student-provided solutions is undecidable in general, the above explanations can be provided efficiently by combining theory for bounded context-free languages [7], canonization of grammars, and grammar transformations.

- **Designing computational reductions.** For teaching reductions, instructors often design tasks for (i) understanding the computational problems involved, (ii) exploring existing reductions via examples, and (iii) designing reductions between computational problems. Task (iii) is a challenge for most students as the design of reductions usually does not follow a straightforward path but requires some creativity by students. When looking for a reduction, one approach by a typical expert is to sequentially try a number of building blocks that they have encountered in the context of other reductions before. An example is provided by the standard reduction from the problem of finding a directed Hamilton path to finding an undirected Hamilton path that transforms a directed graph to an undirected graph by mapping each node 

to a small gadget 

. Constructing such gadgets is one of the typical building blocks when designing reductions.

  One possible approach for supporting instructors in teaching how to design reductions, is to (a) identify and formalize typical building blocks of reductions, (b) develop a simple descriptional language for reductions that allows for combining the building blocks in a simple, modular manner, to (c) study the expressive power of such a language as well as the computational problems arising in finding reductions in such languages, and finally (d) for designing suitable feedback mechanisms for student attempts on constructing reductions. The exploration of (a)–(d) lead to interesting theoretical problems and has already been prototypically implemented (see Figure 5).

An engineering challenge

Building a flexible, extensible, usable, and maintainable teaching support system for formal foundations of computer science is a complex engineering effort which requires, among others, to transfer theoretical results to practice, and to integrate state-of-the-art solutions from a diverse set of domains.

Building a teaching support system is inherently complex and requires the implementation and integration of, among others, modules for providing educational tasks, feedback and advice to students, learning analytics to teachers, interacting with learning management platforms, etc.

There are also challenges specific to teaching support systems for formal foundations, we sketch two of them:

- Algorithms for providing feedback may be known in theory, but may be inefficient in practice and in particular not scale to thousands of users. In our experience, this can often be addressed by employing dedicated SAT solvers and by building custom solutions, e.g., for handling symmetries.

- Helping students to learn how to translate between natural language and formal language — typically a first step when attacking real world problems with formal methods — has been avoided so far in most teaching support systems as it requires to integrate natural language processing. While modern large language models are powerful enough for educational tasks for bridging this natural-formal language gap, a lot of data, engineering, and fine-tuning is required. The main challenge is that a teaching support system should provide correct feedback with very high probability. Figure 7 shows a prototype for an educational task where students can specify propositional variables and their meaning in natural language.

A CS education research challenge

Helping students with a teaching support system requires to understand (among others) students' learning behaviour and motivation, as well as common misconceptions and difficulty-generating factors for formal foundations of computer science.

Didactical aspects of advanced teaching support systems for formal foundations of computer science have mostly been ignored in research. For most concepts taught at university level courses, few didactic foundations have been laid and almost no quantitative and qualitative studies have been done. As a result, there are few guidelines – for example, what common misconceptions students have and how to overcome them, or which factors determine the difficulty of exercises – that can be given to designers of teaching support systems.

Example research questions that have not been addressed for formal foundations of computer science in higher-education contexts are:

(i) How do teaching support systems affect students' learning behaviour and motivation?

(ii) How can teaching support systems be set up to be most effective for heterogeneous groups of students?

(iii) What misconceptions and difficulty-generating factors hinder student success in formal foundations of CS?

(iv) Do (personalized) interventions increase the impact and use of the teaching support system?

Answering these and similar research questions requires a tight integration of expertise in CS education research, educational psychology, formal foundations of computer science, and in building teaching support systems. Data provided by teaching support systems such as Iltis is essential.

Acknowledgments

Many people are contributing to the Iltis project in various ways. Without them, the system would not exist in its current form. The implementation of the system has been supported by Sven Argo, Shrutarv Awasthi, Ariane Blank, Lukas Dienst, Gaetano Geck, Alina Ignatova, Tristan Kneisel, Alexandra Latys, Artur Ljulin, Johannes May, Jan Michalak, Sebastian Peter, Lukas Pradel, Lars Richter, Patrick Roy, Jonas Schmidt, Cedric Siems, Daniel Sonnabend, Fynn Stebel, Felix Tschirbs, Patrick Wieland, Oskar Wilke, and Jan Zumbrink.

The design of the user interface was supported by Christine Orhan. The creation of the course material has been supported by Jill Berg, Tom-Felix Berger, Jan Eyll, Alicia Gayda, Elias Radkte, Mira Schwartz, Dennis Stanglmair, Marco Wojtek, and Cara Volbracht.

Florian Schmalstieg, Thomas Schwentick, Christopher Spinrath, and Nils Vortmeier are providing advice for many aspects of the project.

The example on how to provide extensive feedback for modelling with context-free languages is from joint work of Marko Schmellenkamp and Thomas Zeume with Sven Argo, Cedric Siems, Fynn Stebel, and Sandra Kiefer. The example on how to provide interactive exercises on computational reductions is from joint work of Fabian Vehlken and Thomas Zeume with Julien Grange and Nils Vortmeier. The research questions for CS education research are from a joint project of Thomas Zeume with Philipp Döbler and Jakob Schwerter.

This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

References

[1] Der Iltis ist das Tier des Jahres 2024. `https://www.pronatura.ch/de/tier-des-jahres-2024-iltis`. Accessed: 2024-02-15.

[2] Andrea L. Beach, Charles Henderson, and Noah Finkelstein. Facilitating change in undergraduate STEM education. *Change: The Magazine of Higher Learning*, 44(6):52–59, 2012.

[3] Loris D'Antoni, Martin Helfrich, Jan Křetínský, Emanuel Ramneantu, and Maximilian Weininger. Automata tutor v3. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification – 32nd International Conference, CAV 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2020.

[4] Gaetano Geck, Artur Ljulin, Sebastian Peter, Jonas Schmidt, Fabian Vehlken, and Thomas Zeume. Introduction to Iltis: an interactive, web-based system for teaching logic. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018*, pages 141–146. ACM, 2018.

[5] Gaetano Geck, Christine Quenkert, Marko Schmellenkamp, Jonas Schmidt, Felix Tschirbs, Fabian Vehlken, and Thomas Zeume. Iltis: Teaching logic in the web. *CoRR*, abs/2105.05763, 2021.

[6] Gesellschaft für Informatik e. V. Empfehlungen für Bachelor- und Master-Programme im Studienfach Informatik an Hochschulen. `https://gi.de`, 2016.

[7] Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

[8] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA, 2013.

[9] Susan R. Singer, Natalie R. Nielsen, and Heidi A. Schweingruber. Discipline-based education research. *Washington, DC: The National Academies*, 2012.

[10] The learning tools interoperability protocol. `https://www.1edtech.org/standards/lti`. Accessed: 2024-02-24.

The Distributed Computing Column

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

This month, the Distributed Computing Column is featuring Siddhartha Jayanti, winner of the 2023 Principles of Distributed Computing Doctoral Dissertation Award. His thesis presented a multitude of important concurrent algorithms, including the first scalable algorithm for concurrent union-find, algorithms for concurrent fast arrays, new abortable queue locks and recoverable queue locks, and many more. He defined a new fundamental problem known as "generalized wake-up," whose hardness yields new insights on the work needed for a variety of basic objects. And the thesis contains many more results than can be easily summarized in this short paragraph!

This column focuses on one specific part of his thesis: verifying linearizability in concurrent systems. It is a well-known fact that designing correct concurrent algoroithms is incredibly difficult and bugs are rampant. The standard technique for proving that a concurrent data structure is correct is showing that it is *linearizable*. Unfortunately, that too can be quite challenging! In this article, Siddhartha Jayanti develops a new technique for proving linearizability using only "forward reasoning" techniques: there is no need to reason about the future when analyzing the data structure. This technique yields machine-verifiable proofs, and has been applied to a variety of complex wait-free data structures, including snapshot objects and union-find objects.

Overall, then, this column raises the exciting possibility of simple machine-verified proofs for concurrent data structures, and a future containing more correct concurrent data structures!

The Distributed Computing Column is particularly interested in contributions that summarize recent exciting results, propose interesting new directions, or summarize important open problems in areas of interest. If you would like to write such a column, please contact me.

POSSIBILITY TRACKING: A SIMPLE TECHNIQUE FOR MACHINE-VERIFYING LOCK-FREE DATA STRUCTURES

Siddhartha Jayanti
Google Research, USA

1 Introduction

The multicore revolution has ushered in an era of multiprocessor dominance in computing, however, designing efficient concurrent algorithms with iron-clad guarantees of correctness has remained notoriously hard. While a deterministic single-process algorithm has exactly one possible run, due to asynchrony, even a t step concurrent algorithm with just two processes has 2^t possible runs depending on how the steps of the processes interleave. When we consider algorithms with an infinite horizon, even a deterministic concurrent algorithm has uncountably many possible infinite runs. Designing algorithms that are correct in all of these executions is a grueling task, and programmers often fail to account for some of these executions, leading to subtle and dangerous bugs, known as *races*. Races are pernicious, since they can easily be missed in testing but have harsh consequences when deployed in practice. For example:

- **Mars Rover:** a priority inversion bug in its concurrent code crashed the Pathfinder Rover days after its deployment on Mars and jeopardized the entire multi-million dollar NASA space mission Jones [2013].

- **Northeast Blackout of 2003:** a race in the power grid's energy management system stalled the alarm system for an hour, by which time it was too late to stop a cascading electrical outage that affected an estimated 55 million people across eight states of the USA and the province of Ontario, Canada Poulsen [2004].

- **Therac-25:** the software of the radiation therapy machine, Therac-25, suffered from races that caused it to administer radiation doses that were over

[0]Author information: Google Research, Cambridge, MA, USA. *Email:* sjayanti@google.com

a hundred times as potent as the intended dose, which caused the deaths of at least three people and several more injuries Leveson and Turner [1993]; Lim [1998].

Examples of errors in published concurrent data structures are also not left wanting Colvin and Groves [2005]; Doherty [2003]. These illustrations show just how fatal the consequences can be when multiprocessor code is incorrect, and point to the critical need for concurrent algorithms to be furnished with rigorous, machine-verified guarantees of correctness.

1.1 Which algorithms require rigorous, machine-verified guarantees of correctness?

Two foremost guiding principles in the design of software technology are *modularity* and *top-down design*. Together, these principles state that larger applications should be broken down into smaller well-specified components, i.e., methods, data structures, and simpler algorithms, and that these modular components should be developed independently and made efficient so they can in-turn be called and used in several high-level applications. A strength of this prevalent design strategy is that each core modular component can be built and developed in just one place, and the same well-built component can be used freely in a range of applications today and even the unforeseen applications of tomorrow.

The flip-side of this advantage is a corresponding failure mode, which I term *error proliferation*. Namely, if a core component, such as a data structure, has an uncaught error, it runs the risk of being used in innumerable applications and ultimately crashing critical systems. Even if the component was developed with a low-risk application in mind, the principles of software design make it very easy for the same component to later be integrated into a critical application. Thus, all fundamental data structures and algorithms should be considered critical, and rigorous proofs of correctness are indispensable. Since, concurrent data structures and algorithms are particularly hard to reason about, and important execution schedules and subtle races are often missed in pen-and-paper proofs about them, machine-verified guarantees of correctness are critical for them.

In summary, emphasizing machine-certified correctness of low-level modular components, such as data structures, enhances the reliability of several critical high-level applications.

1.2 Concurrent data structures, linearizability, and future dependence

In this column, I focus on machine-verifiable proofs of correctness for concurrent data structures; particularly, on proving *linearizability*. Linearizability Herlihy and Wing [1990] is the long-standing gold standard for concurrent data structure correctness, and it states that data structure operations must appear to take place *atomically*, i.e., instantaneously, at some point between their invocation and return, even in the face of adversarial asynchronous scheduling. The instant in time at which an operation appears to take effect is called its *linearization point*, and the process is said to have *linearized* its operation after that point in time.

Linearizability is a powerful abstraction, since it allows for efficient software implementations which appear atomic even in the face of tremendous concurrency without requiring global data structure locks—solutions include intricate implementations that use fine-grained locking and lightening-fast implementations that are *lock-free* or even *wait-free* Herlihy [1991]. Linearizability also facilitates composability: its horizontal composability property (known as *locality*), allows algorithmists to prove individual implementations correct without worrying about their interactions with other objects; its vertical composability property allows implementors to replace atomic objects with linearizable implementations.

The most intuitive approach for proving linearizability is via *forward reasoning* methods, where the prover reasons about a concurrent data structure by relating its behavior to that of an atomic reference object as time moves forward. In particular, in a *forward simulation* proof Jonsson [1991], the prover keeps a copy of an atomic reference object and performs an induction over the steps of an arbitrary run of an algorithm using the implemented object, and shows that its behavior is identical to that of the algorithm run with the atomic reference object if the reference object performs operations at the linearization points of implemented object. Forward simulation is easiest when each operation linearizes at a particular line in the operation's code, but this proof technique can work even if an operation's linearizes at different lines of code for different calls, or even if a process's operation can linearize at a step of another process executing a different operation. However, forward simulation proofs are only possible when linearization points can be determined only by looking at the past and present, i.e., for the subclass of so called *strongly linearizable* objects Golab et al. [2011].

There are innumerable examples of linearizable objects however, whose linearization points are *future-dependent*. These implementations have the surprising characteristic that every run can be linearized, but the linearization points of operations can depend on what happens in the future. Such linearizable algorithms are traditionally thought of as notoriously hard to machine-verify, since the intuitive forward simulation proof technique cannot be employed on them Jayanti et al.

[2024]. In particular, as the prover inducts over the run, he cannot know when the linearization points occur (since they are future dependent), and therefore cannot simulate the atomic reference object transitions at the time of the linearization points.

Over the past few decades, researchers have furnished some of these future-dependent algorithms with machine-verified proofs using techniques including backward simulation Jonsson [1989], prophecy variables Abadi and Lamport [1991], partial-order maintenance Khyzha et al. [2017], and aspect-oriented proofs Henzinger et al. [2013]. However, each of these techniques is either well known for being complex and unintuitive for algorithmists, or is incomplete and thus requires ad hoc use. Backward simulation is difficult for algorithmists since it requires reasoning backwards in time Vafeiadis [2008]. Prophecy variables require predicting the future, and are often cited as being "difficult to use in practice" Lamport and Merz [2022]. Partial-order maintenance is known to be incomplete Oliveira Vale et al. [2023]; Jayanti et al. [2024]. Finally, aspect-oriented proofs require new theory to develop the aspects of each data type, and thus only a handful of data types are known to be amenable to such proofs Henzinger et al. [2013]; Dodds et al. [2015]; Öhman and Nanevski [2022]. In particular, a simple, sound and complete technique for proving linearizability has eluded researchers until recently.

1.3 A simple, forward reasoning proof technique for linearizability

In the remainder of this column, I describe the *possibility tracking* (a.k.a. *tracking*) technique for proving the linearizability of concurrent data structures Jayanti [2022]; Jayanti et al. [2024]. This technique was originally described in my doctoral dissertation Jayanti [2022] and subsequently published at this year's *ACM Symposium on Principles of Programming Languages (ACM POPL)* Jayanti et al. [2024].

Possibility tracking is universal, sound, and complete. Universality means that tracking can be applied to any data type; soundness means that a data structure implementation can be proved correct by tracking only if it is linearizable, and completeness means that any linearizable implementation can be proved so using tracking. In addition, tracking is simple and intuitive for algorithmists, since it relies only on forward reasoning, and has been used to produce machine-verified proofs of linearizability for future-dependent and widely-used data structures.

The foundation of our idea lies in replacing the single atomic reference object in the forward simulation technique with a set of such atomic reference objects. In particular, we observe that a run of an algorithm exercising a linearizable object may have several possible linearizations. Each of these linearizations may correspond to a different set of linearization points. While forward simulation maintains just a single atomic reference object, which corresponds to a single possible linearization (i.e., a single possible set of linearization points); our strategy maintains an atomic reference object corresponding to every possible linearization. Since each atomic reference object corresponds to a possible linearization, we call the maintained reference objects *possibilities*. Our proof technique tracks these possibilites over the length of a run, so we call it possibility tracking.

Tracking has been used to give machine-verified proofs of efficient wait-free data structures, including Jayanti's single-writer single-scanner snapshot Jayanti [2005] and the Jayanti-Tarjan union-find objects Jayanti and Tarjan [2016]; Jayanti et al. [2019]; Jayanti and Tarjan [2021], which are used in Google's open-source graph-mining library to enable "parallel clustering algorithms which scale to graphs with tens of billions of edges" Google-Graph-Mining-Team [2023], are the fastest algorithms for computing connected components of large graphs on CPUs Dhulipala et al. [2020] and GPUs Hong et al. [2020], and are employed in several other applications in machine learning Yu et al. [2023]; Wang et al. [2020]; Tseng et al. [2021], graph analysis Shi et al. [2023]; Dhulipala et al. [2020], and program analysis Bloemen et al. [2016].

1.4 Overview

In the remainder of this column, I will describe the tracking method, demonstrate its use in generating a machine-verified proof with a case study of the Herlihy-Wing queue Herlihy and Wing [1987, 1990]—which is notorious in the verification community for its nuanced, future-dependent linearization structure Jung et al. [2019]—and end with some concluding remarks and directions of interest.

A note on proving strong linearizability A variant of our technique, known as Partial Possibility Tracking, is a universal, sound, and complete proof technique for strong linearizability. In fact, our machine-verified proof of the Jayanti-Tarjan union-find objects shows that they are not just linearizable, but strongly linearizable. While I will not say more about strong linearizability in this column, I refer the interested reader to the following reference Jayanti et al. [2024].

2 The Possibility Tracking Proof Technique for Linearizability

To explain how our method works, let $\mathcal{T}$ be any data type, and O be an implementation of type $\mathcal{T}$ for a set Π of processes. To verify that O is linearizable, we augment O with an auxiliary variable $\mathcal{P}$, which helps track *all* possible linearizations of O. In this augmented implementation, which we shall refer to as O^*, $\mathcal{P}$ is a set of *possibilities*. Each possibility $p \in \mathcal{P}$ is a pair (σ, f). In particular, we define O^* such that a possibility (σ, f) is in $\mathcal{P}$, if and only if, there is a linearization of the run until now which corresponds to O's state being σ, and π's state being $f(\pi)$, for each process $\pi \in \Pi$. That is, for each process $\pi \in \Pi$, $f(\pi)$ states whether π has an ongoing operation on O and if it does, whether that operation has linearized and if it has, what the associated response is. More specifically, $f(\pi)$ holds one of three types of values—$(\perp, \perp, \perp)$, $(op, arg, \perp)$, or (op, arg, res)—with the following meaning. If $f(\pi) = (\perp, \perp, \perp)$, π has no ongoing operation on the implementation O. In the other two cases, π has an ongoing operation op on O with argument arg, i.e., π invoked $op_\pi(arg)$, but the operation has not returned yet. Furthermore, if $f(\pi) = (op, arg, \perp)$, π's operation has not yet linearized and if $f(\pi) = (op, arg, res)$, π's operation has linearized (i.e., has taken effect) with a response of res (but the operation has not yet returned to the caller).

We initialize $\mathcal{P}$ to the singleton set $\{(\sigma_0, f_0)\}$, where σ_0 is the initial state of O and, for all $\pi \in \Pi$, $f_0(\pi) = (\perp, \perp, \perp)$, to reflect that there are no ongoing operations on O, initially. Whenever any process π executes a step, the set $\mathcal{P}$ is updated using the following simple rules:

1. Update on operation invocation: If π calls a method on O to invoke an operation $op(arg)$, each possibility $(\sigma, f) \in \mathcal{P}$ is updated from $f(\pi) = (\perp, \perp, \perp)$ to $f(\pi) = (op, arg, \perp)$, to reflect that $op(arg)$ is invoked, but has not yet linearized.

   Notationally, we denote this transformation to the set of possibilities by $Invoke(\mathcal{P}, \pi, op, arg)$.

2. Update on operation return: If π returns from a method by executing a '**return** r' statement, for each possibility $(\sigma, f) \in \mathcal{P}$, if $f(\pi) = (op, arg, \perp)$ or if $f(\pi) = (op, arg, res)$ and $res \neq r$, then (σ, f) is removed from $\mathcal{P}$; on the other hand, if $f(\pi) = (op, arg, res)$ and $res = r$, then $f(\pi)$ is updated to $(\perp, \perp, \perp)$. The removal in the former case ensures that those atomic configurations that do not reflect what happens in the actual execution are filtered out. The update to $(\perp, \perp, \perp)$ in the latter case reflects that π no longer has an ongoing operation.

   Notationally, we denote this transformation to the set of possibilities by $Filter(\mathcal{P}, \pi, r)$.

3. Update on any step: When π executes any step of a method, $\mathcal{P}$ is updated to reflect the possibility that *any* subset of unlinearized ongoing operations may now linearize in *any* order. Accordingly, suppose that $(\sigma, f) \in \mathcal{P}$ before

π takes the step, k is any non-negative integer, $\pi_1, \pi_2, \ldots, \pi_k$ are distinct processes, $f(\pi_1), f(\pi_2), \ldots, f(\pi_k) = (op_1, arg_1, \perp), (op_2, arg_2, \perp), \ldots, (op_k, arg_k, \perp)$. Furthermore, suppose that, by the specification of the data type $\mathcal{T}$ of O, $r_1, r_2, \ldots, r_k$ are the responses if the operations

$$op_1(arg_1), op_2(arg_2), \ldots, op_k(arg_k)$$

are applied in that order, starting from state σ, and σ' is the state after all operations are applied. Then, after the step, (σ', f') appears in $\mathcal{P}$, where f' is the same as f except that

$$f'(\pi_1), f'(\pi_2), \ldots, f'(\pi_k) = (op_1, arg_1, r_1), (op_2, arg_2, r_2), \ldots, (op_k, arg_k, r_k)\,.$$

Notationally, we denote this transformation to the set of possibilities by $Evolve(\mathcal{P})$.

A key observation is that at any time t, the set $\mathcal{P}$ contains an atomic configuration (σ, f) if and only if (σ, f) is consistent with *some* legal linearization up to time t. Intuitively, the "only-if" part of the observation is ensured by the second rule which removes a possibility from $\mathcal{P}$ as soon as there is evidence that the atomic configuration is not consistent with a legal linearization of the history. The "if" part is ensured by the third rule which adds to $\mathcal{P}$ *all* possibilities that are consistent with legal linearizations of the history.

Our main theorem is an immediate consequence of the above observation, and it states that any algorithm run on implementation O^* satisfies the invariant $\mathcal{P} \neq \emptyset$ if and only if the implementation O is linearizable. Equivalently, since the *generator algorithm* $\mathcal{A}$ (see Figure 1) exercises an object implementation to produce all of its possible behaviors by repeatedly making idle processes call arbitrary operations, we see that O is linearizable if and only if $\mathcal{P} \neq \emptyset$ is an invariant of $\mathcal{A}(O^*)$.

Each process $\pi \in \Pi$ is assigned the program $main_\pi()$.

```
    program main_π()
a:      while (true):
b:          choose $(op, arg) \in \{(o, a) \mid o \in \mathcal{T}.OP, a \in \mathcal{T}.ARG_o\}$
            and execute $O.op_\pi(arg)$
```

Figure 1: *Generator algorithm* $\mathcal{A}(O)$ for a set of processes Π, which generates all behaviors of an implemented object O of type $\mathcal{T}$.

Theorem 2.1. *Let O be an implementation of an object of type $\mathcal{T}$ initialized to state σ_0 for a set of processes* Π*, O is linearizable* if and only if $\mathcal{P} \neq \emptyset$ *is an invariant of* $\mathcal{A}(O^*)$.

The theorem gives rise to the possibility tracking verification technique: to verify that an implementation O is linearizable, augment it with the auxiliary variable $\mathcal{P}$ to derive O^* as described, and verify that $\mathcal{P} \neq \emptyset$ is an invariant of $\mathcal{A}(O^*)$. If $\mathcal{P} \neq \emptyset$ is an invariant of $\mathcal{A}(O^*)$, then the theorem implies O is a linearizable implementation of $\mathcal{T}$ and, conversely, if O is a linearizable implementation of $\mathcal{T}$, then the theorem implies that $\mathcal{P} \neq \emptyset$ is an invariant of $\mathcal{A}(O^*)$. Hence, the method is sound and complete. Since the method applies regardless of the data type of the implemented object, it is universal.

3 Case Study: proving the Herlihy-Wing queue

To demonstrate the proof process, I describe our case study of proving the Herlihy-Wing queue Herlihy and Wing [1987, 1990]. I will present the queue implementation in the next subsection, and then explain how we obtained a machine-verified proof of its correctness using possibility tracking.

I chose the Herlihy-Wing queue as the case study, since the algorithm is both short and well known, yet it is notorious in the verification community for being difficult to prove because of its nuanced, future-dependent linearization structure Jung et al. [2019].

Since the Herlihy-Wing queue is nuanced, it is intellectually challenging for a prover to wrap his head around why the algorithm is linearizable. Developing intuition for why the algorithm is linearizable is an inherent part of the proof process, and no proof technique, including possibility tracking can help fully overcome that. Once a prover has understood the intuition for why the queue is linearizable however, possibility tracking makes it easy to translate the intuition into a linearizability proof—even a machine-verified proof; all the prover needs to do is to encode his understanding as a simple algorithmic invariant. That is what I hope to demonstrate to the reader through this case study.

3.1 The Herlihy-Wing Queue Implementation

The Herlihy-Wing queue (see Figure 2) maintains an infinite array of *slots*, i.e., $A[0, 1, 2, \ldots]$, which are initially all *empty*, containing the value $\perp$. The shared counter X stores the value of the next empty slot in A, initially 0. To Enqueue an element v_π, process π atomically fetches and increments X (Line 1) to *claim* the next available slot i_π for its enqueue, and simply *places* its element in the claimed slot, thereby *filling* that slot with its element (Line 2) and returns *ack* (Line 3). To Dequeue, process π reads the value ℓ_π of the counter X (Line 4), which stores the number of claimed slots, and loops through each index j_π of the array A from 0 to $l_\pi - 1$, *checking* each slot $A[j_\pi]$ and *grabbing* the element in the slot if the slot is non-empty (Line 5). If π successfully grabs an element, then it returns it (Line 6). Otherwise, if it reaches the end of the loop, π simply tries again.

Note that a dequeue operation only ever returns with an element; that is, it keeps running indefinitely if the queue is empty.

Base Objects:

- X is a read/F&INC register initialized to 0.
- $A[0,1,2,\ldots]$ is an infinite array, where for each $i \in \mathbb{N}$, $A[i]$ is initialized to $\perp$.

```
    procedure $O$.ENQUEUE$_\pi(v_\pi)$
1:      $i_\pi \leftarrow$ F&INC($X$)
2:      $A[i_\pi] \leftarrow v_\pi$
3:      return $ack$
```

```
    procedure $O$.DEQUEUE$_\pi()$
4:      $\ell_\pi \leftarrow X$
5:      if $\ell_\pi = 0$ then goto 4 else $j_\pi \leftarrow 0$
        $x_\pi \leftarrow$ SWAP($A[j_\pi], \perp$)
        if $x_\pi = \perp$ then
            if $j_\pi = \ell_\pi - 1$ then goto 4
            else {$j_\pi \leftarrow j_\pi + 1$; goto 5}
6:      return $x_\pi$
```

Figure 2: Herlihy-Wing queue implementation Herlihy and Wing [1990]. Each numbered line in this implementation O has at most one shared memory instruction, and is performed atomically. The operation $F\&Inc(var)$ is the atomic fetch-and-increment operation, which returns the current value of var and increments it by one. The operation $Swap(var, new)$ is the atomic swap operation, which returns the current value of var and updates its value to new.

Since each enqueue claims a unique slot, and the slots are claimed in order, at first glance, it is tempting to think that the abstract state of the queue will be a tuple of elements order as in the array A. However, this is not the case, since there can be an arbitrary delay between the time an enqueue claims its slot on Line 1 to the time that it actually fills the slot with its element at Line 2. In particular, dequeuers that are looping through the array will go past slots that have been claimed but have not yet been filled, so elements in slots with higher indices can be grabbed before those in lower indices. Furthermore, since dequeuers can be poised at various different parts of the array (i.e, can have different j values), having looped past slots that had been claimed but not yet filled, the order in which elements are dequeued depends heavily on the order in which processes take steps, thus making the linearization of enqueues highly future-dependent. This future-dependence makes the Herlihy-Wing queue notoriously difficult to prove.

3.2 Proving the linearizability of the Herlihy-Wing queue

To show that the implementation O is linearizable via the possibility tracking technique, we must show that the statement $\mathcal{I}_L \equiv (\mathcal{P} \neq \varnothing)$ is an invariant of $\mathcal{A}(O^*)$, where O^* is the possibility tracker presented in Figure 3. We will prove $\mathcal{I}_L$'s invariance by induction over the length of an arbitrary run. $\mathcal{I}_L$ holds in the initial configuration by the tracker definition, so the base case is straightforward. $\mathcal{I}_L$'s validity in subsequent configurations however, relies not only on its validity in the current configuration, but also on the design of the algorithm, i.e., other invariants of the algorithm that capture the states of the various program variables. Thus, in order to go through with the induction, we must strengthen $\mathcal{I}_L$ to a stronger invariant $\mathcal{I}$ that meets two conditions: (a) $\mathcal{I}$ is *inductive* and (b) $\mathcal{I}$ implies $\mathcal{I}_L$.

For most proofs, identifying the strengthened inductive invariant $\mathcal{I}$ is the real intellectual challenge. Once the correct $\mathcal{I}$ is identified, its actual proof by induction tends to be elementary. The statement of the inductive $\mathcal{I}$ for the Herlihy-Wing queue is presented in Figure 4. Since $\mathcal{I}_L$ is a conjunct of $\mathcal{I}$, proving $\mathcal{I}$'s invariance immediately implies $\mathcal{I}_L$'s invariance, and thus that the Herlihy-Wing queue is linearizable.

Understanding the Inductive Invariant

The invariant may appear long, but a closer examination reveals that almost all of the conjuncts—all but $\mathcal{I}_P$—are fairly elementary. $\mathcal{I}_T$ states that slots that are yet to be claimed remain empty. $\mathcal{I}_U$ states that different enqueuing processes claim different slots in the array. $\mathcal{I}_2$ states that a claimed slot remains empty before the process that claims it fills it. $\mathcal{I}_{2,3}$ states that an enqueuer's claimed slot will have an index between 0 and X. Finally, $\mathcal{I}_5$ states that the loop-index j_π will always lie in the loop-interval $[0, \ell_\pi)$ and that the upper loop boundary ℓ_π will never exceed X.

Our core insight about the algorithm is $\mathcal{I}_P$, which identifies a set of possibilities P that must be in the set of tracked possibilities $\mathcal{P}$. The identification of this subset P of "interesting" possibilities is thus the key to understanding why the Herlihy-Wing queue is linearizable. To elucidate this core insight, I explain the definition of P below. Before explaining P, I will explain some preliminaries.

A key insight about the linearization structure of the Herlihy-Wing queue is that while Enqueue operations can linearize at several places in the interval of time between when they grab their slot to when they fill it, a Dequeue operation can always be thought of as linearizing at the moment that it successfully grabs the element that it will return.

Helpful to defining the set of interesting possibilities P, is the function *val*, which maps array indices in $\mathbb{N}$ to their corresponding values. Formally, given an

Base Objects:

- X is a read/F&Inc register initialized to 1.
- $A[0, 1, 2, \ldots]$ is an infinite read/write/Swap array, where each $A[i]$ is initialized to $\perp$.
- $\mathcal{P}$ initialized to $\{(\sigma_0, f_0)\}$ is a meta-configuration, where σ_0 is the empty sequence, and f_0 maps each process $\pi \in \Pi$ to $(\perp, \perp, \perp)$.

```
procedure $O^*$.Enqueue$_\pi(v_\pi)$
    $\mathcal{P} \leftarrow Invoke(\mathcal{P}, \pi, \text{Enqueue}, v_\pi)$
1:      $i_\pi \leftarrow$ F&Inc$(X)$
        $\mathcal{P} \leftarrow Evolve(\mathcal{P})$
2:      $A[i_\pi] \leftarrow v_\pi$
        $\mathcal{P} \leftarrow Evolve(\mathcal{P})$
3:      return $ack$
        $\mathcal{P} \leftarrow Filter(\mathcal{P}, \pi, ack)$
```

```
procedure $O^*$.Dequeue$_\pi()$
    $\mathcal{P} \leftarrow Invoke(\mathcal{P}, \pi, \text{Dequeue}, \perp)$
4:      $\ell_\pi \leftarrow X$
        $\mathcal{P} \leftarrow Evolve(\mathcal{P})$
5:      if $\ell_\pi = 0$ then goto 4 else $j_\pi \leftarrow 0$
        $x_\pi \leftarrow$ Swap$(A[j_\pi], \perp)$
        if $x_\pi = \perp$ then
            if $j_\pi = \ell_\pi - 1$ then goto 4
            else $\{ j_\pi \leftarrow j_\pi + 1;$ goto 5$\}$
        $\mathcal{P} \leftarrow Evolve(\mathcal{P})$
6:      return $x_\pi$
        $\mathcal{P} \leftarrow Filter(\mathcal{P}, \pi, x_\pi)$
```

Figure 3: Tracker O^* for the queue implementation O for processes Π presented in Figure 2

index i, $val(i)$ is defined as: the value of $A[i]$ if $A[i]$ is non-empty, the value of v_π if π has claimed index i, but is yet to fill it, and $\perp$ otherwise.

To understand the definition of P, we ask ourselves the question: *What are the possible states of the queue at any point in time?* We break this question into two parts: understanding which elements are in the queue, and understanding what order they are in.

Firstly, it is clear that the elements in the queue must be of the form $val(i)$ for some indices i. Thus, the set of elements in the queue must be $\{val(i) \mid i \in I, val(i) \neq \perp\}$ for some subset of indices $I \in [0, X)$. Since Enqueue operations linearize by the time they fill their claimed slot, elements that have been placed in the array but have not yet been grabbed must be in the queue, thus $A[i] \neq \perp \implies i \in I$. Elements corresponding to slots that have been claimed but not yet filled may or may not be in the queue, since the corresponding Enqueue operation may or may not have linearized.

After choosing a possible subset of indices I, we get to the what order $\alpha \in Perm(I)$ the corresponding elements $val(i)_{i \in I}$ occupy in the queue state. This is where we make the most incisive insight. We observe that for a given permutation

α, there is a possibility p with state $p.\sigma = val(\alpha_1), \ldots, val(\alpha_{|\alpha|})$ if α is a $Justified$ permutation. Here, we define the predicate $Justified(\alpha)$ to hold if and only if: for every two indices $\alpha_m, \alpha_n \in I$ that appear at the mth and nth position of the permutation α, where $m < n$: either $\alpha_m < \alpha_n$—i.e., the indices are not inverted in the permutation order—or if they are inverted and the former slot $A[\alpha_n]$ is filled, then there must be a dequeing process π that has checked past the smaller index α_n (i.e., $\alpha_n < j_\pi$) in its checking-loop whose upper index ℓ_π exceeds the larger index α_m. In mathematical notation:

$$\begin{aligned} Justified(\alpha) \;\triangleq\;& \forall m, n \in [1, |\alpha|] : (\alpha_m < \alpha_n) \lor (A[\alpha_n] \neq \bot \\ & \implies \\ & \exists \pi \in \Pi : pc_\pi = 5 \land \alpha_n < j_\pi \land \alpha_m < \ell_\pi) \end{aligned}$$

Finally, since I is the set of indices corresponding to Enqueue operations that have linearized, we know that any pending enequeuers that have linearized are exactly those whose claimed location's indices appear in I. Likewise, since we know that all Dequeue operations linearize at the last iteration of Line 5, we know that a pending dequeuer π has linearized if and only if $pc_\pi = 6$. Putting all of these insights together yields the definition of the set of interesting possibilities P, as defined in Figure 4.

The invariant $\mathcal{I}_P$ simply states that this set of interesting possibilities P is non-empty and indeed contained in the set of possibilities $\mathcal{P}$.

Machine-verified proof

With the strengthened invariant $\mathcal{I}$ in hand, the actual induction proof is quite mechanical, making it a perfect fit to be checked and certified by a machine. The proof of the induction step has six cases, one for each line of the implementation, and it comprehensively justifies why each of the invariant conjuncts holds after an arbitrary process π whose program counter currently points to a line l executes that line of code. Our inductive proof of $\mathcal{I}$ is an invariant of $A(\mathcal{O}^*)$ has been checked by the TLA+ Proof System (TLAPS) Jayanti et al. [2023b].

4 Related Work

The Linearizability correctness condition for concurrent shared-memory data structures was introduced in a landmark paper by Herlihy and Wing in 1990 Herlihy and Wing [1990]. In the ensuing three decades, a tremendous amount of research has focused on proving linearizability, using various different methods, including: refinements Lamport [1983], forward simulation Jonsson [1991], backward

$$\mathcal{I} \equiv \mathcal{I}_L \wedge \mathcal{I}_P \wedge \mathcal{I}_T \wedge \mathcal{I}_U \wedge \mathcal{I}_2 \wedge \mathcal{I}_{2,3} \wedge \mathcal{I}_5$$

In this expression, the various conjuncts on the right hand side are defined below.

- $\mathcal{I}_L \equiv \mathcal{P} \neq \emptyset$
- $\mathcal{I}_P \equiv P \subseteq \mathcal{P} \wedge P \neq \emptyset$
- $\mathcal{I}_T \equiv \forall i \in \mathbb{N} : i \geq X \implies A[i] = \bot$
- $\mathcal{I}_U \equiv \forall \pi, \pi' \in \Pi : \pi \neq \pi' \wedge pc_\pi, pc_{\pi'} \in \{2, 3\} \implies i_\pi \neq i_{\pi'}$
- $\mathcal{I}_2 \equiv \forall \pi \in \Pi : pc_\pi = 2 \implies A[i_\pi] = \bot$
- $\mathcal{I}_{2,3} \equiv \forall \pi \in \Pi : pc_\pi \in \{2, 3\} \implies 0 \leq i_\pi < X$
- $\mathcal{I}_5 \equiv \forall \pi \in \Pi : pc_\pi = 5 \implies 0 \leq j_\pi < \ell_\pi \leq X$

- $\forall i \in \mathbb{N} : val(i) \triangleq \begin{cases} A[i], & \text{if } A[i] \neq \bot \\ v_\pi, & \text{if } \exists \pi \in \Pi : pc_\pi = 2 \wedge i_\pi = i \\ \bot & \text{otherwise} \end{cases}$

- $Justified(\alpha) \triangleq$
  $\forall m, n \in [1, |\alpha|] : (\alpha_m < \alpha_n) \vee (A[\alpha_n] \neq \bot$
  $\implies$
  $\exists \pi \in \Pi : pc_\pi = 5 \wedge \alpha_n < j_\pi \wedge \alpha_m < \ell_\pi)$

- $$P \triangleq \left\{ p \;\middle|\; \begin{array}{l} \exists I \subseteq [0, X), \exists \alpha \in Perm(I) : \\ \quad \forall i \in I : val(i) \neq \bot \wedge \\ \quad \forall i \in [0, X), A[i] \neq \bot \implies i \in I \wedge \\ \quad Justified(\alpha) \wedge \\ \quad p.\sigma = val(\alpha_1), \ldots, val(\alpha_{|\alpha|}) \wedge \\ \quad \forall \pi \in \Pi : \\ \qquad pc_\pi \in \{1, 2, 3\} \implies p.f(\pi).op = \textsc{Enqueue} \wedge p.f(\pi).arg = v_\pi \\ \qquad pc_\pi \in \{4, 5, 6\} \implies p.f(\pi).op = \textsc{Dequeue} \wedge p.f(\pi).arg = \bot \\ \qquad pc_\pi = 3 \vee (pc_\pi = 2 \wedge i_\pi \in I) \implies p.f(\pi).res = ack \wedge \\ \qquad pc_\pi = 6 \implies p.f(\pi).res = v_\pi \wedge \\ \qquad pc_\pi \notin \{3, 6\} \implies p.f(\pi).res = \bot \end{array} \right\}$$

Figure 4: Invariant $\mathcal{I}$ of $\mathcal{A}(O^*)$, where O^* is the implementation of the queue tracker in Figure 3.

simulation Jonsson [1989], forward-backward simulation Lynch and Vaandrager [1995], history and prophecy variables Owicki and Gries [1976]; Abadi and Lamport [1991], aspect-oriented proofs Henzinger et al. [2013], partial-order maintenance Khyzha et al. [2017], and the use of several proof-logics such as interval temporal logic Schellhorn et al. [2011], separation logic Jung et al. [2019], and category theory based methods Oliveira Vale et al. [2023]. The various techniques differ in range of applicability, mechanization, simplicity of use, scope for modularity and several other quantitative and qualitative metrics. The full body of work is too large to cover in a related work section like this one, but I make an attempt to cover some central ideas here. Several of the well-established techniques are mentioned in Dongol and Derrick's survey paper Dongol and Derrick [2014].

A significant portion of linearizability proofs are simulation proofs. A simulation proof incrementally relates the behavior of an implementation to the behavior of an abstract specification: a *forward simulation* does so in the natural direction of execution, while a *backward simulation* does so in the reverse direction. Forward simulation involves only forward reasoning and is thereby among the most intuitive methods for proving linearizability. Traditionally, forward simulation has been used to prove the linearizability of data structures with fixed linearization points Abdulla et al. [2017]; Vafeiadis [2009]; Amit et al. [2007]. Schellhorn et al. proved that backward simulation is a universal, sound, and complete proof technique for linearizability verification Schellhorn et al. [2014], and gave a mechanized proof in KIV Reif et al. [1998] of the correctness of the Herlihy-Wing queue, which is notorious for its future-dependent linearization points. Backward simulation, however, is not a silver bullet. Backward simulation proofs are famously complex and are generally unintuitive to algorithm designers since they require reasoning about the execution of the algorithm in reverse Vafeiadis [2008]; Dongol and Derrick [2014]; Khyzha et al. [2017]. The simulation techniques can also be combined in a forward-backward simulation Lynch and Vaandrager [1995]; Colvin et al. proved the linearizability of Heller et al.'s concurrent list-based set implementation using this technique Colvin et al. [2006]; Heller et al. [2006]; their proof is verified by the PVS proof system.

Some recent works extend forward simulation like techniques to produce pen-and-paper proofs of linearizability of more complex data structures through the use of "commitment points". In particular, Khyzha et al. give a proof technique that maintains a partial order over operations, such that all total orders that respect the partial order are valid linearizations Khyzha et al. [2017]. This maintenance of a partial order allows them to be more tolerant to future-dependence than traditional forward simulation methods which maintain a single total order. In particular, as the future unfolds, the technique makes the partial order stricter at *commitment points* to eliminate total orders that are no longer linearizable. Khyzha et al. give pen-and-paper proofs for the Herlihy-Wing queue, time-stamped queue, and an optimistic set. Bouajjani et al. similarly extend forward simulation techniques to show some queues with fixed linearization points for dequeue and some stack data structures can be proved using forward simulation like methods using commitment points and partial orders Bouajjani et al. [2017]. These authors provide pen-and-paper proofs for the Herlihy-Wing queue and a time-stamped stack data structure. Both these works extend the scope of forward reasoning methods beyond data structures with fixed linearization points. However, the commitment points method with a partial order is not complete Oliveira Vale et al. [2023]; Jayanti et al. [2024].

An alternative to simulation based proofs are proofs using history and prophecy variables. History variables, a.k.a. auxiliary variables, remember the past Owicki and Gries [1976], while prophecy variables foresee/predict the future Abadi and Lamport [1991]. Lynch notes that arguments using history variables alone are akin to forward simulation arguments, while those using prophecy variables alone are akin to backward simulation arguments, and those using a combination of history and prophecy variables are akin to forward-backward simulation arguments Lynch and Vaandrager [1995]. Similarly to backward simulation, prophecy variables also suffer from being "difficult to use in practice" Lamport and Merz [2022]. In the context of our paper, the most related work that uses these variables is the Ph.D. thesis of Vafeiadis Vafeiadis [2008]. In particular, he presents a technique that annotates algorithms with single assignment variables, and stores linearization information into the single-assignment variables with the aid of prophecy variables to help in resolve future-dependent linearization points. He uses the technique to obtain machine-verified proofs of a stack, list, RDCSS (restricted double-compare single-swap Harris et al. [2002]), and MCAS (multiword compare-and-swap). This technique however, is restricted to a class of lock-free algorithms that linearize at CAS operations and another class of read-only methods.

Introduced by Henzinger et al. in 2013, aspect-oriented proofs are non-simulation based techniques that exploit the semantics of particular data types in order to reduce proofs of linearizability to proofs of simpler properties called aspects Henzinger et al. [2013]. The technique is inherently non-universal however, requiring new theory to be developed about the aspects that need to be proved about each data type. Henzinger et al.'s original paper develops the theory for queues, and Chakraborty et al. produced a machine-verified proof of the Herlihy-Wing queue using this method Chakraborty et al. [2015]. The technique was later extended for stacks by Dodds et al. [2015]. Recently, the technique has been extended to snapshot objects by Öhman et al. who have also used it to prove several of Jayanti's snapshot algorithms Jayanti [2005]; Öhman and Nanevski [2022].

Researchers have also explored several specific-purpose program logics for proving linearizability Vafeiadis et al. [2006]; Schellhorn et al. [2011]; Jung et al. [2019]; Oliveira Vale et al. [2023]. Jung et al., in particular, have machine-verified the linearizability of the Herlihy-Wing queue using the Iris framework for separation logic in Coq Jung et al. [2018]. None of these techniques are known to be complete.

In context, our tracking technique is, to our knowledge, the only forward reasoning method to achieve universality, soundness, and completeness.

5 Conclusion and Remarks

In this ongoing era of the multicore revolution, concurrent algorithms are playing a pivotal role in critical systems. The human mind struggles to tackle the complexities of asynchrony, so traditional pen-and-paper proofs—which often gloss over cases or try to capture the high-level at the cost of missing fine details—are often insufficient to fully convince ourselves that key concurrent algorithms are race-free. The lack of rigorous correctness guarantees of concurrent code have often contributed to the failures of consequential systems, such as the Mars Rover failure, the Northeast Blackout of 2003, and the Therac-2 tragedies. All of this evidence points to the importance of machine-verifying concurrent data structures, the key building blocks of concurrent and parallel algorithms.

While a simple, universal, and complete technique for proving the correctness of concurrent data structures has eluded researchers for decades, recent work has broken this barrier. In this column, I explained the *possibility tracking technique* for proving the linearizability of concurrent data structures, and have demonstrated the technique's efficacy by presenting the machine-verified proof of the notriously challenging Herlihy-Wing queue. The technique has also been used to machine-verify efficient and widely used data structures, including Jayanti's single-writer single-scanner snapshot object, and the Jayanti-Tarjan union-find objects. Collectively, these verified algorithms are noted for their complexity, speed, and wide-spread use.

My principle motivation in writing this column is to share my excitement for machine-verification and attaining reliable guarantees of correctness for distributed algorithms. I look forward to machine-verifiying many more algorithms myself, but am also hoping to see machine-verification of algorithms become more mainstream across the distributed computing community.

Due to the inherent complexity of concurrent algorithms, I believe that machine verification can play an even wider role in providing robust, trusted guarantees. My collaborators and I are extending the possibility tracking technique to incorporate other variants of linearizability, such as *strict linearizability* Aguilera and Frølund [2003], *durable linearizability* Izraelevitz et al. [2016], and *recoverable linearizability* Berryhill et al. [2015]; Jayanti et al. [2023a]. We are also designing techniques to verify liveness properties, such as lock- and waitfreedom, and properties of mutual exclusion locks, such as starvation-freedom and first-come-first-served fairness. Finally, we are developing techniques to produce machine-verified proofs of time complexity guarantees of multiprocess algorithms.

While linearizability and its variants have become the gold standard for data structure correctness, there are several algorithms both in the literature and in applications that satisfy weaker consistency guarantees, such as sequential and causal consistency. To my knowledge, universal and complete techniques for these data structures are still wanting, and it would be great to see progress on these important directions.

Acknowledgements

I would like to thank my collaborators on the possibility tracking verification work, particularly, Prasad Jayanti and Ugur Yavuz.

References

M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991. ISSN 0304-3975.

P. A. Abdulla, F. Haziza, L. Holík, B. Jonsson, and A. Rezine. An integrated specification and verification technique for highly concurrent data structures. *International Journal on Software Tools for Technology Transfer*, 19(5):549–563, Oct 2017. ISSN 1433-2787.

M. K. Aguilera and S. Frølund. Strict linearizability and the power of aborting. Technical Report HPL-2003-241, Hewlett-Packard Labs, 2003.

D. Amit, N. Rinetzky, T. W. Reps, M. Sagiv, and E. Yahav. Comparison under abstraction for verifying linearizability. In W. Damm and H. Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 477–490. Springer, 2007.

R. Berryhill, W. M. Golab, and M. Tripunitara. Robust shared objects for non-volatile main memory. In E. Anceaume, C. Cachin, and M. G. Potop-Butucaru, editors, *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

V. Bloemen, A. Laarman, and J. van de Pol. Multi-core on-the-fly SCC decomposition. In *Proceedings of the 21st ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '16, page to appear, 2016.

A. Bouajjani, M. Emmi, C. Enea, and S. O. Mutluergil. Proving linearizability using forward simulations. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*, pages 542–563. Springer, 2017.

S. Chakraborty, T. A. Henzinger, A. Sezgin, and V. Vafeiadis. Aspect-oriented linearizability proofs. *Logical Methods in Computer Science*, Volume 11, Issue 1, Apr. 2015.

R. Colvin and L. Groves. Formal verification of an array-based nonblocking queue. In *10th International Conference on Engineering of Complex Computer Systems (ICECCS 2005), 16-20 June 2005, Shanghai, China*, pages 507–516. IEEE Computer Society, 2005.

R. Colvin, L. Groves, V. Luchangco, and M. Moir. Formal verification of a lazy concurrent list-based set algorithm. In T. Ball and R. B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 475–488. Springer, 2006.

L. Dhulipala, C. Hong, and J. Shun. ConnectIt: A framework for static and incremental parallel graph connectivity algorithms, 2020.

M. Dodds, A. Haas, and C. M. Kirsch. A scalable, correct time-stamped stack. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, page 233–246, New York, NY, USA, 2015. Association for Computing Machinery.

S. Doherty. Modelling and verifying non-blocking algorithms that use dynamically allocated memory. In *Victoria University of Wellington*, 2003.

B. Dongol and J. Derrick. Verifying linearizability: A comparative survey. *CoRR*, abs/1410.6268, 2014.

W. M. Golab, L. Higham, and P. Woelfel. Linearizable implementations do not suffice for randomized distributed computation. In L. Fortnow and S. P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 373–382. ACM, 2011.

Google-Graph-Mining-Team. Google graph-mining. `https://github.com/google/graph-mining`, 2023.

T. L. Harris, K. Fraser, and I. A. Pratt. A practical multi-word compare-and-swap operation. In *Proceedings of the 16th International Conference on Distributed Computing*, DISC '02, pages 265–279, London, UK, UK, 2002. Springer-Verlag.

S. Heller, M. Herlihy, V. Luchangco, M. Moir, W. N. Scherer, and N. Shavit. A lazy concurrent list-based set algorithm. In J. H. Anderson, G. Prencipe, and R. Wattenhofer, editors, *Principles of Distributed Systems*, pages 3–16, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

T. A. Henzinger, A. Sezgin, and V. Vafeiadis. Aspect-oriented linearizability proofs. In P. R. D'Argenio and H. Melgratti, editors, *CONCUR 2013 – Concurrency Theory*, pages 242–256, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1): 124–149, January 1991. ISSN 0164-0925.

M. Herlihy and J. M. Wing. Axioms for concurrent objects. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, Germany, January 21-23, 1987*, pages 13–26. ACM Press, 1987.

M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990. ISSN 0164-0925.

C. Hong, L. Dhulipala, and J. Shun. Exploring the design space of static and incremental graph connectivity algorithms on GPUs. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, September 2020.

J. Izraelevitz, H. Mendes, and M. L. Scott. Linearizability of persistent memory objects under a full-system-crash failure model. In C. Gavoille and D. Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2016.

P. Jayanti. An optimal multi-writer snapshot algorithm. In H. N. Gabow and R. Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory* *of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 723–732. ACM, 2005.

P. Jayanti, S. Jayanti, and S. Jayanti. Durable algorithms for writable LL/SC and CAS with dynamic joining. In R. Oshman, editor, *37th International Symposium on Distributed Computing (DISC 2023)*, volume 281 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:20, Dagstuhl, Germany, 2023a. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

P. Jayanti, S. Jayanti, U. Y. Yavuz, and L. Hernandez Videa. Artifact for "A Universal, Sound, and Complete Forward Reasoning Technique for Machine-Verified Proofs of Linearizability", POPL 2024, Oct. 2023b.

P. Jayanti, S. Jayanti, U. Yavuz, and L. Hernandez. A universal, sound, and complete forward reasoning technique for machine-verified proofs of linearizability. *Proc. ACM Program. Lang.*, 8(POPL), jan 2024. doi: 10.1145/3632924. URL `https://doi.org/10.1145/3632924`.

S. Jayanti, R. E. Tarjan, and E. Boix-Adserà. Randomized concurrent set union and generalized wake-up. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 187–196, New York, NY, USA, 2019. Association for Computing Machinery.

S. V. Jayanti. *Simple, Fast, Scalable, and Reliable Multiprocessor Algorithms*. PhD thesis, Massachusetts Institute of Technology (MIT), Department of Electrical Engineering and Computer Science, November 2022. Code available at: https://github.com/visveswara/machine-certified-linearizability.

S. V. Jayanti and R. E. Tarjan. A randomized concurrent algorithm for disjoint set union. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 75–82, New York, NY, USA, 2016. ACM.

S. V. Jayanti and R. E. Tarjan. Concurrent disjoint set union. *Distributed Comput.*, 34(6):413–436, 2021.

M. Jones. What really happened to the software on the Mars Pathfinder spacecraft? `https://www.rapitasystems.com/blog/what-really-happened-software-mars-pathfinder-spacecraft`, July 2013.

B. Jonsson. On decomposing and refining specifications of distributed systems. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop,* *Mook, The Netherlands, May 29 - June 2, 1989, Proceedings*, volume 430 of *Lecture Notes in Computer Science*, pages 361–385. Springer, 1989.

B. Jonsson. Simulations between specifications of distributed systems. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR '91, 2nd International Conference on Concurrency Theory, Amsterdam, The Netherlands, August 26-29, 1991, Proceedings*, volume 527 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 1991.

R. Jung, R. Krebbers, J.-H. Jourdan, A. Bizjak, L. Birkedal, and D. Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28:e20, 2018.

R. Jung, R. Lepigre, G. Parthasarathy, M. Rapoport, A. Timany, D. Dreyer, and B. Jacobs. The future is ours: Prophecy variables in separation logic. *Proc. ACM Program. Lang.*, 4(POPL), Dec. 2019.

A. Khyzha, M. Dodds, A. Gotsman, and M. Parkinson. Proving linearizability using partial orders. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings 26*, pages 639–667. Springer, 2017.

L. Lamport. Specifying concurrent program modules. *ACM Trans. Program. Lang. Syst.*, 5(2):190–222, 1983.

L. Lamport and S. Merz. Prophecy made simple. *ACM Trans. Program. Lang. Syst.*, 44(2):6:1–6:27, 2022.

N. Leveson and C. Turner. An investigation of the Therac-25 accidents. *Computer*, 1993.

J. Lim. An engineering disaster: Therac-25, 1998.

N. A. Lynch and F. W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.

J. Öhman and A. Nanevski. Visibility reasoning for concurrent snapshot algorithms. *Proc. ACM Program. Lang.*, 6(POPL), Jan. 2022.

A. Oliveira Vale, Z. Shao, and Y. Chen. A compositional theory of linearizability. *Proc. ACM Program. Lang.*, 7(POPL), Jan. 2023.

S. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.

K. Poulsen. Software bug contributed to blackout. *SecurityFocus*, 2004.

W. Reif, G. Schellhorn, K. Stenzel, and M. Balser. Structured specifications and interactive proofs with KIV. *Automated Deduction—A Basis for Applications: Volume II: Systems and Implementation Techniques*, pages 13–39, 1998.

G. Schellhorn, B. Tofan, G. Ernst, and W. Reif. Interleaved programs and rely-guarantee reasoning with ITL. In C. Combi, M. Leucker, and F. Wolter, editors, *Eighteenth International Symposium on Temporal Representation and Reasoning, TIME 2011, Lübeck , Germany, September 12-14, 2011*, pages 99–106. IEEE, 2011.

G. Schellhorn, J. Derrick, and H. Wehrheim. A sound and complete proof technique for linearizability of concurrent data structures. *ACM Trans. Comput. Logic*, 15(4), September 2014.

J. Shi, L. Dhulipala, and J. Shun. Parallel algorithms for hierarchical nucleus decomposition, 2023.

T. Tseng, L. Dhulipala, and J. Shun. Parallel index-based structural graph clustering and its approximation. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1851–1864. ACM, 2021. doi: 10.1145/3448016.3457278. URL `https://doi.org/10.1145/3448016.3457278`.

V. Vafeiadis. Modular fine-grained concurrency verification. Technical Report UCAM-CL-TR-726, University of Cambridge, Computer Laboratory, July 2008.

V. Vafeiadis. Shape-value abstraction for verifying linearizability. In N. D. Jones and M. Müller-Olm, editors, *Verification, Model Checking, and Abstract Interpretation, 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18-20, 2009. Proceedings*, volume 5403 of *Lecture Notes in Computer Science*, pages 335–348. Springer, 2009.

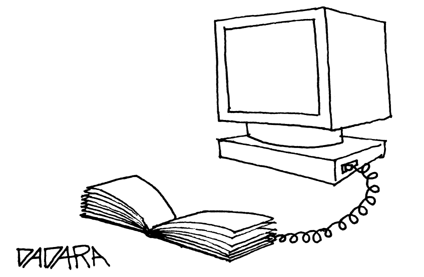
V. Vafeiadis, M. Herlihy, T. Hoare, and M. Shapiro. Proving correctness of highly-concurrent linearisable objects. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '06, page 129–136, New York, NY, USA, 2006. Association for Computing Machinery.

Y. Wang, Y. Gu, and J. Shun. Theoretically-efficient and practical parallel DBSCAN. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of*

Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, pages 2555–2571. ACM, 2020. doi: 10.1145/3318464.3380582. URL `https://doi.org/10.1145/3318464.3380582`.

S. Yu, J. Engels, Y. Huang, and J. Shun. Pecann: Parallel efficient clustering with graph-based approximate nearest neighbor search, 2023.

News and Conference Reports



Report on AFL 2023: 16th International Conference on Automata and Formal Languages Eger, Hungary, September 5–7, 2023

Bianca Truthe
Justus Liebig University Giessen, Germany

The conference series AFL was initiated in 1980. In the first years, it took place every second year, later every third year. It was always organized by a university in Hungary. The most recent conferences of the series were organized in Dobogókő (2005), Balatonfüred (2008), Debrecen (2011), Szeged (2014), and again Debrecen (2017).

This year, the conference was organized by the Institute of Informatics of the University of Szeged and the Faculty of Informatics of the Eszterházy Károly Catholic University of Eger and took place in Eger.

Eger

The invited speakers were

- Galina Jirásková (Slovak Academy of Sciences, Košice, Slovakia) who spoke about 'Operations on Boolean and Alternating Finite Automata',
- Andreas Maletti (University of Leipzig, Germany) who spoke about 'Compositions of Weighted Extended Top-down Tree Transducers', and
- Victor Mitrana (Polytechnic University of Madrid, Spain, and National Institute of R&D for Biological Sciences, Bucharest, Romania) with a talk 'On the Degree of Extension of Some Models Defining Non-Regular Languages'.

Besides the talks by the invited speakers, 18 talks on peer reviewed research papers were presented. They covered many topics in the area of automata and formal languages (fuzzy finite automata, weighted automata, freezing 1-tag systems, 1-limited automata, Horner automata, quantum finite automata, reversible computations, pumping lemmata, tree-controlled grammars, and many more).

On the conference website of the recent edition, you can find a list of all accepted papers:

`https://afl2023.uni-eszterhazy.hu/`

The proceedings were edited by Zsolt Gazdag, Szabolcs Iván, and Gergely Kovásznai and published in the EPTCS series (Electronic Proceedings in Theoretical Computer Science, `https://eptcs.org/`), Volume 386. Extended versions of selected papers will be published in a special issue of the *International Journal of Foundations of Computer Science* (IJFCS).

On top of the Observatory Tower

Besides the scientific sessions, the conference program contained also a social part. It consisted of a visit of the Observatory Tower including an exhibition related to Astronomy and visitors terrasse with a nice view over Eger, a guided tour through the city and into its underworld (caves which were used as wine storage), and a conference dinner in a wine cellary.

Many thanks go to the organizers, program committee members, external reviewers, and participants for the pleasant and successful event. The next edition will probably take place in three years.

Report on CIAA 2023: 27th International Conference on Implementation and Application of Automata Famagusta, North Cyprus, September 19–22, 2023

Bianca Truthe
Justus Liebig University Giessen, Germany

Recently, the 27th edition of the Conference on Implementation and Application of Automata was held in Famagusta (North Cyprus) from September 19 to 22, 2023. The first Workshop on Implementing Automata (WIA'96) was held in London, Ontario, Canada, in 1996. In the year 2000, the workshop WIA became the CIAA conference. The CIAA conference series covers all aspects of implementation, application, and theory of automata and related structures. It aims to attract contributions from both classical automata theory and applications. This year's conference was organized by Benedek Nagy and his colleagues from the Eastern Mediterranean University in Famagusta. It was co-located with the workshop on Non-Classical Models of Automata and Applications (NCMA).

View over Famagusta

The invited speakers were

- Viliam Geffert (University of Košice, Slovakia) who spoke about 'Binary coded unary regular languages',
- Friedrich Otto (University of Kassel, Germany) who gave 'A Survey on Automata with Translucent Letters', and
- Cem Say (University of Istanbul, Turkey) with a talk on 'Finite automata as verifiers'.

Besides the talks by the invited speakers, 20 talks on peer reviewed research papers were presented. They covered many topics in the area of automata theory and beyond (finite automata, push-down automata, transducers, tree automata, hedge automata, verification, Parikh's theorem, pumping lemma, and many more).

On the conference website of the recent edition, you can find the program with all talks:

`https://ciaa.emu.edu.tr/en`

The proceedings were edited by Benedek Nagy and published in the LNCS series by Springer (Lecture Notes in Computer Science), Volume 14151. Extended versions of selected papers will be published in a special issue of the *International Journal of Foundations of Computer Science* (IJFCS).

During the conference trip, in Bellapais

Besides the scientific sessions, the conference provided also a social program. It consisted of a guided tour through the city of Famagusta and the Ghost Town Varosha, a bus trip to Bellapais, and a conference dinner at a restaurant with a beautiful view over the northern coast line and a large variety of delicious meals.

At the end of the conference, Szilárd Zsolt Fazekas invited with a presentation to the next edition of CIAA which is planned to be held next September in Akita (Japan).

Many thanks to the organizers, program committee members, external reviewers, and participants for the pleasant and successful event. We invite all readers of this report to submit papers to CIAA 2024 and to attend the conference.

Report on DCFS 2023: 25th International Conference on Descriptional Complexity of Formal Systems Potsdam, July 4–6, 2023

Bianca Truthe
Justus Liebig University Giessen, Germany

The 25th DCFS took place in Potsdam, Germany, from July 4 to 6, 2023. It was organized jointly by the IFIP Working Group 1.02 on Descriptional Complexity and by the Institute of Computer Science at the University of Potsdam.

Group Photo of Participants in Park 'Sanssouci'

At the conference, 16 scientific talks were given, 3 of them by invited speakers, namely

- Pascal Caron (University of Rouen, France) who spoke about 'Operational state complexity revisited: The contribution of monsters and modifiers',
- Friedrich Otto (University of Kassel, Germany) who gave a talk 'On the influence of the various parameters of the Restarting Automaton on its expressive capacity and descriptional complexity',
- Rogério Reis (University of Porto, Portugal) with a talk titled 'Size matters, but let's have it on average'.

The other contributions (all peer reviewed) were written by 35 authors. All papers are contained in the proceedings, edited by Henning Bordihn, Nicholas Tran, and György Vaszil, and published by Springer as volume 13918 in the series *Lecture Notes in Computer Science*. Full versions of selected papers will be published in a special issue of the journal *Information and Computation*.

On the conference website, you can find the program with all talks:

`https://www.cs.uni-potsdam.de/dcfs2023/`

Besides the scientific sessions, there were two more: the Business Meeting of the IFIP Working Group 1.02 and a Special Session on the occasion of the 25th edition of DCFS where JÃ¼rgen Dassow (Otto-von-Guericke University of Magdeburg, Germany), one of the co-founders of the DCFS conference series, gave a ceremonial address. In the Business Meeting, Martin Kutrib (Justus-Liebig-UniversitÃ¤t Giessen) as the chair of the working group gave an overview about activities of the group as well as the past and future of the conference series DCFS. He also announced that Rogério Reis will soon take over the chair.

The history of the conference series DCFS and other information can be found at the homepage:

`http://www.informatik.uni-giessen.de/dcfs/`

As social events, we had a tour through Sanssouci Park very well prepared and guided by students of the University Potsdam as well as a great conference dinner.

We thank everybody, in particular Henning Bordihn and the local organizers, who made the conference a successful event. The next DCFS will take place in Santa Clara, California, USA, organized by Nicholas Tran. We invite all readers of this report to submit papers to DCFS 2024 and to take part in the conference.

Report on NCMA 2023:
13th International Workshop on Non-Classical Models of Automata and Applications
Famagusta, North Cyprus, September 18–19, 2023

Bianca Truthe
Justus Liebig University Giessen, Germany

The conference venue

The workshop series on Non-Classical Models of Automata and Applications (NCMA) was founded in the year 2009; up to the year 2019, it took place every year. After a pandemic caused pause, there was a restart with a new edition last year where also the organization of this year's event was settled. Now, the 13th edition was held in Famagusta (North Cyprus) on September 18 and 19, 2023. The workshop was organized by Benedek Nagy and his colleagues from the Eastern Mediterranean University in Famagusta. It was co-located with the Conference on Implementation and Application of Automata (CIAA).

The history and other information about the conference series can be found at the homepage which is available here:

`https://www.cs.uni-potsdam.de/NCMA/`

On the workshop website of the recent edition, you can find the program with all talks:

`https://ncma.emu.edu.tr/en`

The invited speakers were

- Friedrich Otto (University of Kassel, Germany) who gave 'A survey on automata with translucent letters' and
- György Vaszil (University of Debrecen, Hungary) with a talk on 'Membrane computing and Petri nets'.

Besides the talks by the invited speakers, 14 talks on peer reviewed research papers were presented. The 11 regular papers and 3 short papers covered many topics in the area of automata theory and beyond (tree-walking-storage automata, forgetting automata, sweeping permutation automata, quantum finite state automata, graph grammars, ordered grammars, signed grammars, contextual grammars, sticker systems, stream cipher and pseudorandom number generators, and many more).

The proceedings were edited by Rudolf Freund and Benedek Nagy and published in the EPTCS series (Electronic Proceedings in Theoretical Computer Science, `https://eptcs.org/`), Volume 388. Extended versions of selected papers will be published in a special issue of *Acta Informatica*.

Besides the scientific sessions, there was one more: A joint session with the workshop on Theoretical Informatics dedicated Victor Mitrana on the occasion of his 65th birthday held at the Faculty of Mathematics and Informatics of the University of Bucharest, organized by Marius Dumitran, Radu Gramatovici, and Florin Manea. During this session, in which both venues were connected via a video conference system, several participants of the NCMA who had been invited to contribute also to the workshop honoured Victor in very personal presentations full of memories of joint research.

This special session was followed by the social program. It consisted of a guided tour to the ancient city of Salamis, a visit of the Monastery St. Barnabas with Icon Museum and a guided tour through the city of Famagusta. Afterwards, a splendid dinner with a huge variety of delicious meals was awaiting us in a restaurant situated directly at the Mediterranean Sea.

Sightseeing in Salamis

Many thanks to the organizers, program committee members, external reviewers, and participants for the pleasant and successful event. The next issue of NCMA is planned to be held next year in Göttingen (Germany), co-located with DLT, organized by Florin Manea. We invite all readers of this report to submit papers to NCMA 2024 and to come to the conference next August.

European Association for Theoretical Computer Science

E A T C S

EATCS

HISTORY AND ORGANIZATION

EATCS is an international organization founded in 1972. Its aim is to facilitate the exchange of ideas and results among theoretical computer scientists as well as to stimulate cooperation between the theoretical and the practical community in computer science.

Its activities are coordinated by the Council of EATCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EATCS. This assembly is scheduled to take place during the annual **I**nternational **C**olloquium on **A**utomata, **L**anguages and **P**rogramming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the "Bulletin of the EATCS;"
- Award of research and academic career prizes, including the EATCS Award, the Gödel Prize (with SIGACT), the Presburger Award, the EATCS Distinguished Dissertation Award, the Nerode Prize (joint with IPEC) and best papers awards at several top conferences;
- Active involvement in publications generally within theoretical computer science.

Other activities of EATCS include the sponsorship or the cooperation in the organization of various more specialized meetings in theoretical computer science. Among such meetings are: CIAC (Conference of Algorithms and Complexity), CiE (Conference of Computer Science Models of Computation in Context), DISC (International Symposium on Distributed Computing), DLT (International Conference on Developments in Language Theory), ESA (European Symposium on Algorithms), ETAPS (The European Joint Conferences on Theory and Practice of Software), LICS (Logic in Computer Science), MFCS (Mathematical Foundations of Computer Science), WADS (Algorithms and Data Structures Symposium), WoLLIC (Workshop on Logic, Language, Information and Computation), WORDS (International Conference on Words).

Benefits offered by EATCS include:
- Subscription to the "Bulletin of the EATCS;"
- Access to the Springer Reading Room;
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from "EATCS Monographs" and "EATCS Texts;"
- Discount (about 70%) per individual annual subscription to "Theoretical Computer Science;"
- Discount (about 70%) per individual annual subscription to "Fundamenta Informaticae."

Benefits offered by EATCS to Young Researchers also include:
- Database for Phd/MSc thesis
- Job search/announcements at Young Researchers area

(1) THE ICALP CONFERENCE

ICALP is an international conference covering all aspects of theoretical computer science and now customarily taking place during the second or third week of July. Typical topics discussed during recent ICALP conferences are: computability, automata theory, formal language theory, analysis of algorithms, computational complexity, mathematical aspects of programming language definition, logic and semantics of programming languages, foundations of logic programming, theorem proving, software specification, computational geometry, data types and data structures, theory of data bases and knowledge based systems, data security, cryptography, VLSI structures, parallel and distributed computing, models of concurrency and robotics.

Sites of ICALP meetings:

- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, UK 1976
- Turku, Finland 1977
- Udine, Italy 1978
- Graz, Austria 1979
- Noordwijkerhout, The Netherlands 1980
- Haifa, Israel 1981
- Aarhus, Denmark 1982
- Barcelona, Spain 1983
- Antwerp, Belgium 1984
- Nafplion, Greece 1985
- Rennes, France 1986
- Karlsruhe, Germany 1987
- Tampere, Finland 1988
- Stresa, Italy 1989
- Warwick, UK 1990
- Madrid, Spain 1991
- Wien, Austria 1992
- Lund, Sweden 1993
- Jerusalem, Israel 1994
- Szeged, Hungary 1995
- Paderborn, Germany 1996
- Bologne, Italy 1997
- Aalborg, Denmark 1998
- Prague, Czech Republic 1999
- Genève, Switzerland 2000
- Heraklion, Greece 2001
- Malaga, Spain 2002
- Eindhoven, The Netherlands 2003
- Turku, Finland 2004
- Lisabon, Portugal 2005
- Venezia, Italy 2006
- Wrocław, Poland 2007
- Reykjavik, Iceland 2008
- Rhodes, Greece 2009
- Bordeaux, France 2010
- Zürich, Switzerland 2011
- Warwick, UK 2012
- Riga, Latvia 2013
- Copenhagen, Denmark 2014
- Kyoto, Japan 2015
- Rome, Italy 2016
- Warsaw, Poland 2017
- Prague, Czech Republic 2018
- Patras, Greece 2019
- Saarbrücken, Germany (virtual conference) 2020
- Glasgow, UK (virtual conference) 2021
- Paris, France 2022
- Paderborn, Germany 2023

(2) THE BULLETIN OF THE EATCS

Three issues of the Bulletin are published annually, in February, June and October respectively. The Bulletin is a medium for *rapid* publication and wide distribution of material such as:

- EATCS matters;
- Technical contributions;
- Columns;
- Surveys and tutorials;
- Reports on conferences;
- Information about the current ICALP;
- Reports on computer science departments and institutes;
- Open problems and solutions;
- Abstracts of Ph.D. theses;
- Entertainments and pictures related to computer science.

Contributions to any of the above areas are solicited, in electronic form only according to formats, deadlines and submissions procedures illustrated at `http://www.eatcs.org/bulletin`. Questions and proposals can be addressed to the Editor by email at `bulletin@eatcs.org`.

(3) OTHER PUBLICATIONS

EATCS has played a major role in establishing what today are some of the most prestigious publication within theoretical computer science.

These include the *EATCS Texts* and the *EATCS Monographs* published by Springer-Verlag and launched during ICALP in 1984. The Springer series include *monographs* covering all areas of theoretical computer science, and aimed at the research community and graduate students, as well as *texts* intended mostly for the graduate level, where an undergraduate background in computer science is typically assumed.

Updated information about the series can be obtained from the publisher.

The editors of the EATCS Monographs and Texts are now M. Henzinger (Vienna), J. Hromkovič (Zürich), M. Nielsen (Aarhus), G. Rozenberg (Leiden), A. Salomaa (Turku). Potential authors should contact one of the editors.

EATCS members can purchase books from the series with 25% discount. Order should be sent to:
Prof.Dr. G. Rozenberg, LIACS, University of Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands
who acknowledges EATCS membership and forwards the order to Springer-Verlag.

The journal *Theoretical Computer Science*, founded in 1975 on the initiative of EATCS, is published by Elsevier Science Publishers. Its contents are mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

The Editor-in-Chief of the journal currently are D. Sannella (Edinburgh), L. Kari and P.G. Spirakis (Patras).

ADDITIONAL EATCS INFORMATION

For further information please visit `http://www.eatcs.org`, or contact the President of EATCS:
Prof. Artur Czumaj,
Email: `president@eatcs.org`

EATCS MEMBERSHIP

DUES

The dues are €40 for a period of one year (two years for students / Young Researchers). Young Researchers, after paying, have to contact `secretary@eatcs.org`, in order to get additional years. A new membership starts upon registration of the payment. Memberships can always be prolonged for one or more years.

In order to encourage double registration, we are offering a discount for SIGACT members, who can join EATCS for €35 per year. We also offer a five-euro discount on the EATCS membership fee to those who register both to the EATCS and to one of its chapters. Additional €35 fee is required for ensuring the *air mail* delivery of the EATCS Bulletin outside Europe.

HOW TO JOIN EATCS

You are strongly encouraged to join (or prolong your membership) directly from the EATCS website `www.eatcs.org`, where you will find an online registration form and the possibility of secure online payment. Alternatively, contact the Secretary Office of EATCS:

Mrs. Efi Chita,
Computer Technology Institute & Press (CTI)
1 N. Kazantzaki Str., University of Patras campus,
26504, Rio, Greece
Email: `secretary@eatcs.org`,
Tel: +30 2610 960333, Fax: +30 2610 960490

If you are an EATCS member and you wish to prolong your membership or renew the subscription you have to use the Renew Subscription form. The dues can be paid via paypal and all major credit cards are accepted.

For adittional information please contact the Secretary of EATCS:

Dmitry Chistikov
Computer Science
University of Warwick
Coventry
CV4 7AL
United Kingdom
Email: `secretary@eatcs.org`,